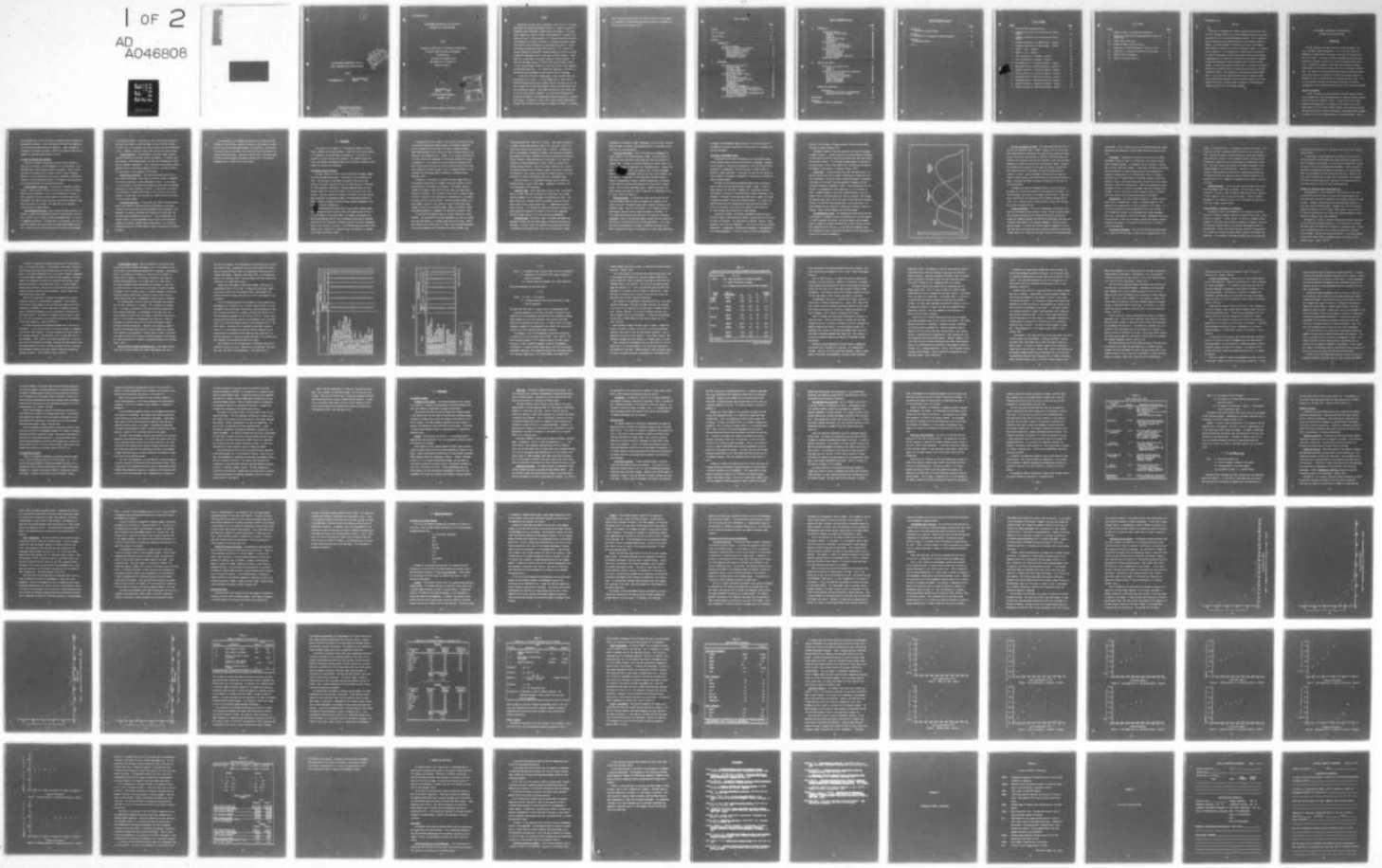
AD-A046 808 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
A PRELIMINARY CALIBRATION OF THE RCA PRICE S SOFTWARE COST ESTI--ETC(U)
SEP 77 J SCHNEIDER

UNCLASSIFIED AFIT/GSM/SM/77S-15 NL

1 OF 2
AD A046808



(1)

A PRELIMINARY CALIBRATION OF THE RCA PRICE S SOFTWARE COST ESTIMATION MODEL

Thesis

John Schneider, IV
Captain USAF

AFIT/GSM/SM/77S-15

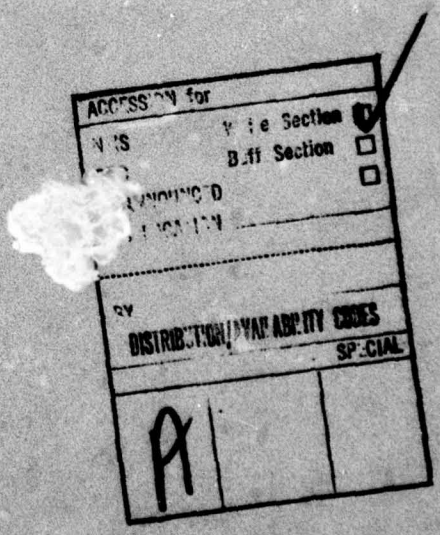
(See 1473)

AFIT/GSM/SM/77S-15

A PRELIMINARY CALIBRATION OF THE RCA PRICE S SOFTWARE COST ESTIMATION MODEL

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science



by

John Schneider, IV
Captain USAF

Graduate Systems Management

September 1977

Approved for public release; distribution unlimited.

PREFACE

During the nine years that I have been in the Air Force, I have been associated with software development projects in a number of capacities: prospective user, programmer, systems analyst, and manager. This experience, supported by similar findings in the professional literature, has convinced me that, given good people, it is the early decisions that most often determine the success or failure of a software development project. These decisions are often made in an environment where there is significant pressure to make the project look attractive. This pressure can only be withstood by professional software managers who have a sound body of empirical knowledge on which to base their project estimates. Today, there are a large number of extremely competent software managers. The empirical knowledge necessary for them to make good estimates, however, is still being developed through the continuing efforts of many competent and dedicated researchers. This research was undertaken in an effort to contribute to the development of that essential body of knowledge.

I would like to take this opportunity to thank the many people who contributed to this effort. Dr. Frank Freiman and Dr. Robert Park of RCA PRICE Systems deserve special thanks, both for allowing me to use the PRICE S model and for their many valuable insights into both PRICE S and cost estimating in general. Many thanks also are due to Ms. Freda Kurtz of the Air Force Avionics Laboratory. Ms. Kurtz helped formulate the project and arranged for the computational resources necessary to complete the research. In addition, I would like to thank the many ASD personnel who took time from their busy schedules to support this effort. Of course,

special thanks go to my advisor, Dr. Charles McNichols, and my reader, Dr. Joseph Cain, without whose understanding support and guidance this effort would not have been possible.

TABLE OF CONTENTS

		Page
Preface		ii
List of Figures		vi
List of Tables		vii
Abstract		viii
I	INTRODUCTION	1
	The Cost of Software	1
	The Need for Software Cost Estimates	2
	Capital Budgeting Decisions	2
	Source Selection Decisions	2
	Project Scheduling	3
	System Design Trade-Off's	3
	Performance Evaluation	3
II	BACKGROUND	5
	The Software System Life-Cycle	5
	Conceptual Stage	7
	Validation Stage	7
	Full-Scale Development	8
	Maintenance Stage	8
	The Software Development Process	9
	Design Phase	10
	The Implementation Phase	10
	The Test and Integration Phase	12
	Systems Engineering	12
	Programming	13
	Documentation	13
	Configuration Management	13
	Current Concepts in Software Cost Estimation	14
	Methods of Estimating Software Development Costs	15
	The Measurement Problem	17
	Factors Affecting Software Development Costs	17
	The Cost of Maintenance	28
	An Introduction to PRICE S	30

TABLE OF CONTENTS (Cont'd)

		Page
III	METHODOLOGY	34
	The Research Problem	34
	Statement of the Problem	34
	Purpose	34
	Objectives	35
	Scope and Limitations	35
	Assumptions	36
	Data Collection	36
	The Initial Interview	36
	Cost/Schedule Data Collection	38
	Statistical Data Collection	39
	Methods of Analysis	43
	Subjective Evaluation	43
	Statistical Analysis	43
	PRICE S Methodology	44
	Sensitivity Analysis Techniques	46
	Systems Descriptions	46
IV	ANALYSIS AND RESULTS	48
	Overview of the Systems Studied	48
	System A	48
	System E	50
	Evaluation of the Data Collection Methodology	51
	The Initial Interview	51
	Cost/Schedule Data Collection	53
	Statistical Data Collection	55
	System A Adjustments	64
	System E Adjustments	64
	Sensitivity Analysis	66
V	SUMMARY AND CONCLUSIONS	76
	Conclusions	76
	Conclusions Relating to the Methodology	76
	Conclusions Relating to PRICE S	77
	Bibliography	79
	Appendix A A Glossary of PRICE S Terminology	81

TABLE OF CONTENTS (Cont'd)

Page

Appendix B
The Initial Interview Format. 83

Appendix C
Derivation of the Sequential Sampling Equation. 88

Appendix D
System Descriptions 94

Vita. 97

LIST OF FIGURES

Figure		Page
1	PRICE S Resource Expenditure Curves	11
2	Frequency Distribution for Instructions per Module - System A.	56
3	Frequency Distribution for Instructions per Module - System E.	57
4	Frequency Distribution for Module Weight - System A	58
5	Frequency Distribution for Module Weight - System E	59
6	TNINST vs. Cost - System A.	67
7	TNINST vs. Cost - System E.	67
8	Total Instructions vs. Schedule - System A.	68
9	Total Instructions vs. Schedule - System E.	68
10	Development Cost vs. Instruction Density - System A	69
11	Development Cost vs. Instruction Density - System E	69
12	Schedule Duration vs. Instruction Density - System A.	70
13	Schedule Duration vs. Instruction Density - System E.	70
14	Development Cost vs. Capacity Utilization - System A.	71
15	Development Cost vs. Capacity Utilization - System E.	71
16	Schedule Duration vs. Capacity Utilization - System A	72
17	Schedule Duration vs. Capacity Utilization - System E	72

LIST OF TABLES

Table		Page
I	Impact of Factors on Software Cost Estimation.	19
II	Exponential Relationships Between Development Effort and System Rate.	23
III	PRICE S Application Types.	41
IV	Summary of Sample Size Calculations.	60
V	Comparison of Alternative Methods of Computing "Mix"	62
VI	Comparison of "Interactive" Modules From Two Systems	63
VII	Baseline PRICE S Parameters.	65
VIII	Sensitivity Analysis Results	74

ABSTRACT

Each year, the Department of Defense spends more than three billion dollars on computer software, yet software managers are notoriously unable to predict the cost of software development projects. This is especially true of preliminary cost estimates made during the formative stages of a project. Even when parametric relationships are used, such estimates depend heavily on anology with previously developed systems. The purpose of this research is to investigate ways of gathering and using descriptive data for the purpose of making preliminary software cost estimates. A methodology for the collection of descriptive information on software systems was developed and used to describe several avionics software systems. The data thus gathered was then used to "calibrate" the PRICE S software cost estimation model by relating particular values of several "subjective" PRICE S input parameters to the observed software system data. It was found that certain characteristics of software systems could be objectively measured, and that the PRICE S model is not incompatible with avionics software systems developed for the Aeronautical Systems Division of Air Force Systems Command.

A PRELIMINARY CALIBRATION OF THE RCA PRICE S SOFTWARE COST ESTIMATION MODEL

I. INTRODUCTION

The past few years have been a period of intensive change in the area of computer software development. This is true both within the Department of Defense (DOD) in particular and within the software community as a whole. The concept of software engineering has gained wide acceptance, and seems to be well on its way to becomming a mature discipline. Concurrently, software professionals have developed a deeper understanding of what software systems are, and how they should be developed. Yet, even as the tools and techniques required to develop software systems have evolved, the inability of software development managers to estimate the cost of a given project has remained notoriously poor. This failure to develop accurate cost estimating techniques has important consequences for both software developers and for their potential customers.

The Cost of Software

In 1976, the Deputy Assistant Secretary of Defense (Material Acquisition) estimated that annual DOD expenditures for computer software exceeded three billion dollars (Gansler, 1976:1). On the civilian side, Boehm estimated that expenditures for software in the United States in 1976 would exceed 15 billion dollars (Boehm, 1975:4). While there may be some overlap and inaccuracy in these two estimates, they unquestionably document the overall size of the software industry in the United States. Even so,

such estimates only reflect the resources consumed in the development and maintenance of software. They do not purport to measure the opportunity costs of resources wasted because of schedule slippages, uneconomical assignment of resources, and general inability to correctly guage the consequences of possible future courses of action.

The Need for Software Cost Estimates

There are a number of reasons why accurate, timely estimates of total cost are essential to the management of all large projects, including those that include computer software development. Some of the key decisions which depend on cost estimates include: capital budgeting decisions, source selection decisions, project scheduling, system design trade-off's, and performance evaluation. The impact of cost estimates on each of these decisions is discussed briefly below.

Capital Budgeting Decisions. The decision to undertake a specific project is usually made on the basis of some kind of comparison between expected costs and benefits. Even if the considerations involved are highly subjective in nature, some estimate of the project duration and the manpower (or other critical resources) required must normally be made before the project is approved. Thus some form of cost estimate is required prior to project approval.

Source Selection Decisions. Once a project has been approved, it is necessary to select the organization that will actually develop the software. This may be an "in-house" software development team, or it may be an independent contractor. In either case, it is necessary to be able to accurately estimate the project costs for differing development options. This is especially true since unrealistically low bids are not uncommon

in the software industry. Such bids may result from technical naivete on the part of the bidder, or from an attempt to "buy in" on the contract. In either case, it is necessary that unrealistically low bids be identified as such. Otherwise a realistic source selection decision cannot be made.

Project Scheduling. The development of detailed project schedules requires the ability to estimate resource requirements. In essence, each work package in the schedule becomes a project, the requirements for which must be estimated independently. Thus, the quality of the project schedule depends directly on the project manager's ability to accurately estimate the requirements of each component of the project.

System Design Trade-Off's. Very few system acquisitions involve a pure software development. Most often, the software system is "embedded" in a larger system which is being developed as a whole. In such cases, it is not unusual to find that certain system functions might be performed equally well by either hardware or software. In such cases, the decision to use hardware or software often hinges on the estimated costs of each approach. In some cases, such decisions can radically alter the architecture of the entire system.

Performance Evaluation. In many cases, the closest available approximation to an objective standard of project performance is the project schedule, and the associated resource requirement (cost) estimates, "Good management" is normally associated with projects that achieve their objectives ahead of schedule and "below cost." Yet, as stated above, the schedule itself is directly dependent on the ability of the responsible managers to accurately estimate resource requirements. Thus, realistic performance evaluation is often directly linked to the quality of resource estimates.

From the above, it is apparent that the ability to obtain realistic estimates of the resources required to complete a given project is necessary for any software development manager to be effective. Unfortunately, the state-of-the-art of software cost estimating has not yet advanced far enough to meet these needs. Therefore, there has been a continuing search for new and better methods of estimating software costs. The research documented here is a small part of that search.

II. BACKGROUND

The purpose of this chapter is to introduce the reader to the RCA PRICE S software cost estimating model. Before this can be done, however, it is necessary to place the model in perspective. To do this, it is necessary to briefly discuss three subjects: the software system life-cycle, the software development process, and the current literature on the subject of software cost estimation.

The Software System Life-Cycle

The term "system life-cycle" is used to describe the stages through which any man-made system passes. The system is first conceived as an idea. Then the idea is transformed into physical reality and used. Finally, the system invariably outlives its usefullness and is discarded. Air Force Regulation (AFR) 800-14, "Computer Resources Acquisition and Support" refers to five discrete stages in the life-cycle of Air Force systems. These are termed the conceptual, validation, full-scale development, production, and deployment stages. Any system may be defined in terms of these five stages including software systems (Department of the Air Force, 1974:2-1 through 2-6).

One point which must be made clear is that a given software system can pass through the entire life-cycle while the "supra-system" of which it is a part remains in one single stage. For example, the software that must be developed and used to test a prototype aircraft may be completely replaced with new "operational" software during the full-scale development phase of the aircraft's life-cycle. The following discussion refers to the stages of the life-cycle of a software system. No reference is intended to the stage of the supra-system.

Although the "official" stages of the system life-cycle may be used as a basis for discussing software systems, the software product has several characteristics which make this frame of reference awkward. These characteristics follow from the fact that a software system is not tangible in any meaningful sense. Rather, it is a logical construct, as is a book, that exists independent of the physical medium that is used to represent it. In a recent Air Force Institute of Technology (AFIT) class, Peterson used this fact to justify two modifications of the life-cycle concept for software systems: lack of a production phase, and the existence of a maintenance phase as opposed to a deployment phase (Peterson, 1976).

Production, in the sense of a system life-cycle stage, refers to the consistent replication of a system in sufficient quantity to allow the objectives of the system to be achieved. For complex, physical systems, this stage is often one of the most costly and complicated in the entire life-cycle. This is simply not the case for software systems. All that is required is that some physical representation of the system be copied, verified, and distributed. Even for extremely complex software systems, the cost of this process is usually insignificant when compared to the cost of the other stages. Hence, for software systems the production phase is, in general, not significant (Peterson, 1976).

Discussing the deployment stage of a software system also creates problems. The connotations of the term deployment are simply not compatible with what happens to a software system during this stage. Deployment connotes assets being "used up," supply lines providing replacement parts, and maintenance facilities repairing "broken" systems. But

software can be neither "used up" nor "broken." What actually happens to a software system during the last stage of its life-cycle is that it is modified on a more or less continuous basis to reflect: corrections for errors, enhancements of capability, changes in the operating environment, and changes in the users' perception of what the system should accomplish. Almost invariably, these modifications are performed, or at least controlled, by a single organization in order to keep all copies of the software system identical. Within the software community, this process of controlled modification of software systems is universally referred to as maintenance. Thus, within the context of software systems, it is more meaningful to discuss a maintenance stage than a deployment stage.

Given the above, it is reasonable to discuss the life-cycle of a software system in terms of four stages: conceptual, validation, full-scale development, and maintenance.

Conceptual Stage. Every system evolves from an idea. The identification of "good" ideas and the expansion of these ideas into a formal, approved statement of user requirements characterizes this phase. AFR 800-14 states that "the major definitive document resulting from this phase is the initial system specification..." (Department of the Air Force, 1974:2-2). It is during this phase that the major decisions concerning what the system will do, where it will fit into the general scheme of things, and how it will work are made and documented.

Validation Stage. During this stage, the system requirements are developed in detail, and the ramifications of acquiring the system are evaluated. In the Air Force. the "authenticated system specification" is developed, and the feasibility, cost, risk, and schedule of system

development are examined in detail (Department of the Air Force, 1974:2-2). When this phase is complete, the developing agency is authorized to proceed with full-scale development.

Full-Scale Development. This is the stage which encompasses the actual creation of the complete software system. For most systems of significant size, this phase involves much more than the production of computer code. In order for the final product to be useful, most or all of the following must be completed in an acceptable manner: computer programs, documentation, user manuals, user training, programmer manuals, programmer training, test plans and procedures, and test results. In addition, for most operational software systems, a number of nonoperational (support) computer programs must be developed to facilitate the testing and maintenance of the operational software system. The full-scale development stage is complete when the using agency accepts the system for routine operational use (Department of the Air Force, 1974:2-2 through 2-3).

Maintenance Stage. The maintenance stage begins during the full-scale development stage and extends throughout the remainder of the system life-cycle. Maintenance begins when a computer program has been completely coded. From this time forward, it will undergo a series of modifications that will end only when the program is discarded. At first these modifications are controlled by the individual programmer (debugging), later by the developing agency (system test and integration), and finally by the operational user (routine operations). In any case the intent of maintenance is the same: to modify the system so as to make it operate more closely in accordance with current system objectives.

In essence, the maintenance stage constitutes a continuing process of refinement which permits the system to grow and adapt to a changing operational environment.

The Software Development Process

During both the full-scale development and the maintenance stages of the software system life-cycle, new software is developed. Depending on the situation, it may be most convenient to view "the software being developed" as encompassing: the entire system, a single program, or possibly a single subroutine. In any case, the idea that the process by which software is developed is independent of the particular problem has strong intuitive appeal.

The model of the software development process which is described below is the one on which the RCA PRICE S model is based. It must be noted, however, that RCA PRICE S Systems, Inc. intentionally leaves a wide degree of flexibility in the definitions of model terminology. This approach is intended to allow individual model users a degree of latitude in adapting the model to their own accounting system. As a result, the specific definitions and interpretations given below are the responsibility of this author, while the overall concept is the creation of Dr. Frank Freiman and his associates at PRICE Systems.

According to Freiman, the software development process can be described in terms of three overlapping phases, during which five activities, or tasks, are performed. The three phases are termed: "design," "implementation," and "test and integration." The five activities are "systems engineering," "programming," "configuration management," "documentation," and "program management." All five of these activities are performed

during all of the phases, although the amount of each activity varies from phase to phase (Freiman, 1977).

The general pattern of resource expenditure for each of the phases is shown in Figure 1. The most significant characteristics of the phases are that they have specific starting and ending points, that they overlap, and that they interact. The interaction between phases is such that if one phase begins or ends at the "wrong" time, the resources that need to be expended in other phases is increased (Freiman, 1977).

Design Phase. The design phase begins when the agency which is to actually develop the software system is authorized to proceed with full-scale development. During this phase, system architecture is determined, requirements are allocated to system components (programs), and the individual programs are designed in detail. Each program moves from the design to the implementation phase separately, as it is "released to code." Proposed changes to the current system baseline are also considered to be in the design phase until they are approved. When changes are approved, they either take on the status of the individual program(s) involved, or they force the program(s) back to the design phase. Of course changes can also create and delete entire programs or groups of programs. Newly created programs enter the design phase.

The Implementation Phase. The implementation phase begins when the first program is "released to code" and continues until the last program is formally accepted for testing. During this phase, each programmer codes and debugs his program, documents the "as built" design, and turns over responsibility for the program to the test and integration team. The implementation phase is generally the smallest of the three phases in terms of total resource expenditure.

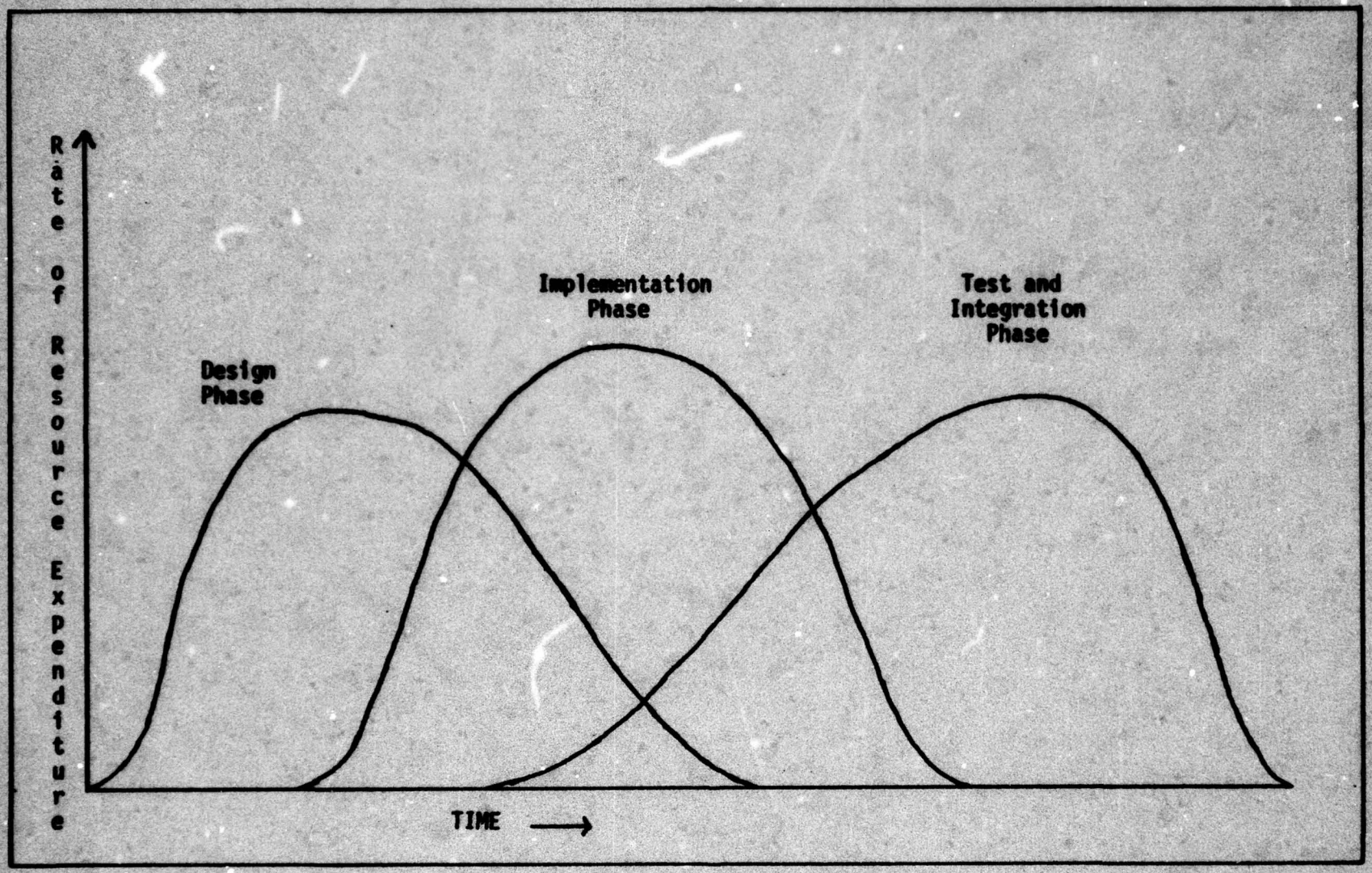


Figure 1. PRICE S Resource Expenditure Curves (Not Drawn to Scale)

The Test and Integration Phase. This phase begins when the core of the test and integration team is formed. Normally, this is at the point where system design has progressed far enough for test planning to being, but well before any formal testing is required. During this phase, two different but inseparable functions are performed. First, the individual programs are fitted into place as part of the overall system. Secondly, the system and each component of the system are tested to insure that each required function is performed, and that the system functions properly as a whole. When a programmer certifies that his program is ready for formal testing, that program enters the testing and integration phase. This phase ends when the system is turned over to the user for routine operations.

Throughout the software development process, five activities are performed, all of which are essential to the success of the project. As with the phases, the dividing line between the activities is rather arbitrary, and different situations may warrant different boundary definitions. Still, the main thrust of each activity is quite clear and they may be easily discussed in turn.

Systems Engineering. This activity encompasses the technical tasks which are concerned with the system as a whole. During the design phase, this includes developing precise system specifications, dividing the system into component programs and defining the associated interfaces, and allocating requirements to the separate components. During the implementation phase, it includes the review of specific components to insure that they meet system objectives and a continuing evaluation of the system through analysis and simulation to minimize the risk of not meeting system

requirements. Finally, during the test and integration phase the systems engineering task encompasses system oriented testing and problem resolution.

Programming. Programming is the central activity on any software development project, in that it is the activity that directly creates working computer programs. It encompasses the work necessary to design, code, and test individual programs. In the areas of designing and testing, where the distinction between systems engineering and programming is necessarily rather obscure, the distinguishing criteria is whether a particular task is oriented toward an individual program or toward the system as a whole. During the design phase, the majority of the programming work is individual program design. During the implementation phase, the emphasis shifts to individual programmers coding and debugging their programs. Finally, during the test and integration phase, programming activities include formal testing and maintenance programming.

Documentation. The primary tangible product of a software development project is documentation. Indeed, it sometimes seems as if the generation of printed material is the sole purpose of the project. If all that was required for the documentation activity was drafting, it could probably be included in the programming and systems management activities. Unfortunately, this is not the case. Documentation must not only be drafted, it must also be edited, printed, distributed, reviewed, corrected, and updated. For projects of any size, this is a task of significant magnitude.

Configuration Management. This activity involves the determination, at all times, of precisely what is, and is not, an approved part of the

system. To accomplish this, it is necessary to perform three tasks. The first task involves incorporating specifications into the "system baseline." This may be viewed as formally declaring a particular document as being "officially correct." Once a document has been incorporated into the baseline, changes may only be made through the configuration control task. This taks involves the evaluation of alleged deficiencies in and proposed changes to the system baseline. Finally, it is necessary to provide for the dissemination and control of approved baseline material. The conflicting needs for both easy access to baseline information and positive control of all changes to the baseline often severely complicate this task.

Program Management. As with any other organized human effort, software development projects must be managed. This activity includes the supervisory, financial, legal, and general administrative tasks necessary to plan, organize, direct and control the project. Ideally, program management is the integrating activity that coordinates the other activities into a single, coherent effort.

Current Concepts in Software Cost Estimation

According to Clapp, "cost estimation can be defined as predicting the cost of resources needed to carry out a process which delivers a specified set of products." (Clapp, 1976:5) In the case of computer software systems, the resources consist largely of manpower and computer time costs. Of course the relevant process is the software development process described above. Finally, the product generally consists of "operational" (i.e., tested and formally accepted) computer software and software documentation. It is customary to estimate separately such associated costs

as supplies and travel as well as additional tasks such as user training. (Clapp, 1976:6)

In most cases, a cost estimate should include all of the relevant costs that would be incurred under the assumed conditions throughout the entire life-cycle of the system. For all practical purposes this includes the costs of full-scale development and maintenance. Most of the work in the field of software cost estimation has focused on full-scale development, while maintenance has been undeservedly neglected. Recently, however, some preliminary work has been done on maintenance cost estimation. This work will be discussed briefly following a more complete survey of the state-of-the-art in the full-scale development area.

Methods of Estimating Software Development Costs

Although there is little agreement in the literature on the specific cost mode which should be used to estimate software development costs, there does seem to be general agreement on the component methods which may be used. In general, most software cost estimating models combine at least some of the following methods: decomposition, historical analogy, parametric equations, and unit of work. Other methods, such as the use of resource constraints are not so much cost estimating methods as upper bounds for "design to cost" type situations.

Decomposition is simply a method of breaking the problem into pieces so that each may be estimated separately. It is often used to estimate the total number of instructions or modules in a proposed system. Alternatively, decomposition may be used to break the system into individual "work packages," which may be estimated separately. In any case, decomposition almost always requires assumptions about the structure of the proposed system. (Clapp, 1976:15)

If there is a universally recognized method of estimating software cost, it is historical analogy. In this method, the estimate is based on the historical cost experience associated with one or more similar projects. If no similar experience exists, the proposed system is decomposed to the point where analogies may be drawn for each component. The advantages of this method are simplicity and potential accuracy when a good analogy can be found. On the other hand, there is a serious danger of making invalid analogies, and the cost of maintaining and searching the required historical records must be considered. (Nelson, 1966; Aron, 1969; Smith, 1975; Clapp, 1976)

Most of the recent work on software cost estimation has revolved around the search for reliable parametric equations. Such equations relate cost to other parameters which describe the system, and which are "more easily" estimated. The most common type of parametric equation used for software development is based on the number of instructions in the proposed system. This method is extremely fast and simple to apply. It is also only as accurate as the available equations and the estimates of the input parameters. (Clapp, 1976:16-17)

The final basic method of estimating software cost is the unit of work method. In this case, the task of building the system is decomposed into discrete "work packages." Each work package must be small enough to be accurately estimated. Aron mentions a limit of eight weeks work for one programmer. (Aron, 1969:8) The system development cost is then the sum of the costs for each work package. Both Aron and Smith maintain that this method works well for small systems, but breaks down for large software development projects with their complex interactions between both programs and people. (Aron, 1969:8-9; Smith, 1975:2-18)

The Measurement Problem. Before proceeding to a discussion of the factors which affect software development costs, it is necessary to discuss the way in which software development cost is measured. Unfortunately there is, at present, no concensus as to what should be included as a relevant cost of software. This problem is especially acute when the software development project is an integral part of a larger effort. In such cases, systems oriented functions may be performed, in large part, by central facilities whose costs are not directly allocable to software development. Such situations, as well as normal differences in accounting procedures between organizations introduce a wide variability into recorded software costs that is independent of actual resource consumption.

This problem seems to be more serious for government cost analysts than for their counterparts in industry. This is because each company may, if it chooses, standardize its own cost accounting procedures and cost estimating techniques. The government, on the other hand, must accept software cost data from a wide range of individual accounting systems if it is to establish a comprehensive cost data base. Likewise, government cost analysts must be able to work with proposals based on different accounting assumptions. Because of this problem, a number of proposals have been made which would set uniform standards for software costs reported to the government. The most recent of these is the excellent report by Graver which contains a detailed treatment of the problems and issues associated with attempts to standardize software cost reporting. (Graver, 1977)

Factors Affecting Software Development Costs. Measurement problems aside, one of the major reasons why software development costs are so

difficult to estimate is the large number of factors which have a significant effect on them. Researchers in this area have consistently found that at least 40 different factors can significantly affect the cost of software development. (Weinwurm, 1965; Heard, 1977) To illustrate the extent of this problem, Table I lists the factors which were found to be significant in a recent study by Herd.

Despite the large number of significant factors, recent work has concentrated on only a few of them. Of these, the single most significant factor is the size of the software system. Unfortunately, the precise relationship between size and cost is still being debated in the literature.

The first problem associated with attempts to relate system size to development cost is measuring size. The most common solution is to use the number of delivered object instructions as the basic measure of size. This is usually justified as an attempt to minimize the effects of different programming languages on size. (Graver, 1977:6-8) To date, Herd's group has published the most extensive and conceptually valid investigation of the relationship between system size and development cost in the public domain. The results of his regressions show nearly identical values for R^2 and standard error when two different measures of size were analyzed in conjunction with the same cost data. (Herd,1977) At present, consistency in using a particular definition of size seems to be more important than the particular definition chosen.

Once a measure of size has been chosen, a functional form must be selected to relate it to cost. Historically, two functional forms have been used, the linear and the exponential. The linear form is:

TABLE I

Impact of Factors on Software Cost Estimation

Factor	Software Application				
	Command and Control	Scientific	Business	Utility	All
Communication	H	H	H	H	H
Constrained, CPU Time	M	H	M	H	M
Constrained, Program Memory Size	M	M	M	L	M
Constrained, Time and Memory	H	H	H	H	H
Cost of Secondary Resources	N	N	N	N	N
Cost/Schedule Control Systems Criteria	N	N	N	N	N
Data Management Techniques	M	M	M	M	M
Data Collection, Amount and Method of	N	N	N	N	N
Developer's First Time on Specified Computer	H	H	H	H	H
Developer Using Another Activity's Computer	M	M	M	M	M
Development of Hardware, Concurrent	H	H	M	M	M
Development and Target Computer Different	H	L	M	M	M
Development Personnel Mix	M	M	M	M	M
Development Site	M	M	M	M	M
Development Site, Number of	M	M	M	M	M
Design Complexity	H	H	L	L	M
Design Stability	H	H	H	H	H
Instruction, Definition of	H	H	H	H	H
Innovation, Degree of	H	H	H	H	H
Programmer Testing	H	H	H	H	H
Programming Environment	H	H	H	H	H
Programming Facilities	H	H	H	H	H
Programming Techniques, Modern	H	H	H	H	H
Requirements, Language	H	H	H	H	H
Requirements, Maintainability	H	H	H	H	H

TABLE I (Cont'd)

Factor	Software Application				
	Command and Control	Scientific	Business	Utility	All
Requirements Changes, Operational	H	H	H	H	H
Requirements Definition, Operational	M	H	N	N	L
Requirements/Design Interface, Operational	H	H	H	H	H
Requirements, Quality	H	H	H	H	H
Requirements, Reliability	M	M	M	M	M
Requirements, Special Display	L	L	M	N	L
Requirements, Testing Including V&V	M	M	M	M	M
Requirements, Transportability	M	M	M	M	M
Requirements, User Considered	M	M	M	M	M
Sites Multiple Software Utilization	M	M	M	M	M
Sizing Error	H	H	H	H	H
Software Development Schedule	H	H	H	H	H
Support Software Availability	H	H	H	H	H
Specified Response Time	M	H	H	N	M
Target CPU Designation	H	H	H	H	H
Work Breakdown Structure	H	H	H	H	H

(Herd, 1977:178-179)

LEGEND
H = High significant impact M = Medium significant impact L = Low significant impact N = Negligible significant impact

$$C = aS$$

where: C = a measure of cost (usually either dollars or man-months)

S = a measure of size (usually either number of object or source instructions)

a = a constant (usually estimated using linear regression)

While the Exponential form may be written:

$$C = aS^b$$

where: C, S, and a are as above,

b = a constant exponent (also usually the result of some form of regression)

Of course the linear form is a special case of the exponential form.

The prime arguments for the linear form seem to be simplicity and compatibility with standardized, multivariate linear regression programs. These arguments are not to be discounted, since they do allow other factors to be considered along with size. As long as there is a strong conceptual argument for the exponential form, however, any results that depend on the linear form must remain open to question.

The argument in favor of the exponential form can be traced back at least as far as Farr and Nanus' work in 1964. (Farr, 1964:13) The most prominent proponent of this viewpoint, however, has been Brooks, who makes a strong, conceptual argument in favor of an exponential relationship. In essence, Brooks argues that as the size of the system development team grows, the overhead associated with intra-team coordination increases exponentially. For large systems, this communication

problem becomes, according to Brooks, the dominant influence on development cost. (Brooks, 1975)

Until very recently, the available data concerning the proper value of the exponent was inconclusive. Recently, however, Herd has published an empirical study of software cost estimation which sheds considerable light on this question. This study has two distinct advantages over previous work. First, the data base used (129 cases after deletions) is the largest that this author is familiar with. (Herd, 1977:122) In addition, this is the only study that this author has seen which used nonlinear regression techniques.

Most studies of the exponential relationship have used "log-linear" regression techniques. According to Draper and Smith, this approach is only valid if the logarithm of the error term is normally distributed. (Draper, 1966:132) In the case of software cost data, this does not appear to be a valid assumption. In addition, the magnitude of the errors is large enough to significantly distort the "fit." (Herd, 1977:123)

Herd divided his sample into four types of systems: command and control, scientific, business, and utility. He then used nonlinear regression techniques to solve for the constant parameters. Table II summarizes the results of this analysis. As the table indicates, the exponents calculated vary considerably for different types of systems.

Herd's results are interesting, but still not conclusive. For one thing, the size of the data sample is still quite small considering the dispersion of the data. Another problem is that the largest system included in the sample apparently represented less than 60,000 source

TABLE II

Exponential Relationships Between Development Effort and System Rate

Regression Model: $MM = aI^b$

Where: MM = Total man-months for system development

I = Total instructions in system

a,b = Constants calculated using nonlinear regression

System Type	Type of Instructions	a	b	R^2	Standard Error
Command and Control	OBJECT	4.57	1.29	.78	41.1
	SOURCE	4.09	1.26	.80	41.1
Scientific	OBJECT	4.50	1.07	.74	72.1
	SOURCE	7.05	1.02	.78	72.1
Business	OBJECT	2.90	.78	.48	12.4
	SOURCE	4.50	.78	.61	10.7
Utility	OBJECT	12.04	.72	.30	58.1
	SOURCE	10.08	.81	.45	51.7
All	OBJECT	4.79	.99	*	62.2
	SOURCE	5.26	1.05	*	50.7

*Not reported

(Herd, 1977:124-164)

lines, while most of the data represented much smaller systems. Thus, Herd's results may not adequately reflect "large" software development projects.

Even so, Herd's analysis does seem to be the most extensive and conceptually correct available. Despite its limitations, it presents strong evidence that diseconomies of scale (large systems cost more per instruction) may exist for at least some classes of software system developments. Such evidence should not be ignored, especially considering the large number of small systems represented in the sample.

Unfortunately, size alone is insufficient to explain even half of the variance in software development cost. (Herd, 1977) Some of the other factors which have received significant attention in the literature include application mix, difficulty, schedule constraints, and source language. Each of these is discussed briefly below.

There is evidence that some types of applications are easier to code than others. Herd's classification of systems into four types (command and control, scientific, business and utility) represents one method of modeling this effect. His results seem to support the hypothesis that, however he discriminated between types, there were significant differences between them. (Herd, 1977) This, however, is not the only method of modeling the effects of differences between applications.

Wolverton achieves essentially the same effect by decomposing each system into six categories of routines. His categorization includes: control, pre or post algorithm processors, logical or mathematical algorithms, data management, and time critical processors.

(Wolverton, 1972:9) This approach is based on the assumption that different productivity rates are inherently associated with different types of routines. Under this assumption, different "kinds" of systems would be typified by different "mixes" of these "application types." Thus, Herd's findings are not incompatible with this approach.

Wolverton's approach has several distinct advantages over the method used by Herd. First, it should be possible to define rather simple decision rules for categorizing individual routines. Given these rules, any system can be described. On the other hand, it is not hard to imagine the other approach leading to continuing classification problems. The other major advantage of Wolverton's approach is that it is essentially numerical in nature, and thus quite compatible with quantitative analysis. The other approach is more qualitative in nature, and thus harder to analyze.

Even allowing for different application mixes, the fact remains that some programming tasks are more difficult than others. Even though measuring difficulty is inherently a subjective process, differences in difficulty must be accounted for. Aron has proposed a method of measuring difficulty which uses three discrete levels: easy, medium, and hard. According to this categorization, "easy" programs have very few interactions with other system elements. "Medium" programs, on the other hand, not only have more interactions with other system elements, they are also characterized by the ability to solve general classes of problems, (e.g., language compilers). Finally, "hard" programs interact with many system elements. They are typified by system monitors and operating systems. (Aron, 1969:12-13)

Wolverton has proposed that an additional factor be taken into account when attempting to measure difficulty. In his opinion, "new" (or unfamiliar) applications are harder than "old" (or familiar) applications. Thus, Wolverton advocates the use of six categories, combining Aron's three classifications with the prefix "old" or "new." (Wolverton, 1972:13-14)

Another factor which has a significant effect on software development cost is schedule. Unquestionably, the most eloquent treatment of the relationship between cost and schedule is Brooks' classic essay "The Mythical Man-Month." His discussion is based on the premise that manpower and schedule are directly interchangeable only insofar as the tasks involved are independent of each other. If the tasks involved are inherently sequential, however, adding manpower cannot reduce the time required to complete them. In fact, Brooks argues convincingly that the assignment of too many people to a development organization is counterproductive, because of the increasing level of coordination required of each individual. The result of this argument is Brooks' Law: "Adding manpower to a late software project makes it later." (Brooks, 1975)

Unfortunately, little empirical work has been publsihed on the relationship between cost and schedule. Putnam has developed a software cost model which incorporates some of the relationships required by Brooks' theory. This model is, however, derived from a theoretical, rather than an empirical base. (Putnam, 1969) Herd's group addressed the relationship between system size and development time, but could find no quantitative data with which to measure the cost impact of deviation from an optimum schedule. (Herd, 1977:43-45) Graver's group, on the

other hand, attempted a truly unique analysis of the interrelationships between phases of development. Unfortunately, lack of data severely limited the usefullness of their results. Still, their approach should be of value to future investigators. (Graver, 1977)

The final factor affecting software development cost which will be discussed here is programming language. Developments which use higher order languages (HOL) such as FORTRAN are generally felt to be less expensive than similar developments which use machine oriented languages (MOL). First, each source statement in a HOL program is expanded into several machine instructions, while MOL statements generally correspond directly to individual machine instructions. In addition, it is often argued that systems written in a HOL are easier to read and understand. (Graver, 1977:6-10)

Graver's group has recently investigated the effects of language on development cost. Using log-linear regression, they found that developments using MOL's tended to require about twice as many total man-months as developments that used HOL's. The statistical confidence level calculated for this relationship was not high enough (0.8), however, to permit a definitive conclusion to be drawn. Graver does report, however, that these results agree with the results of some controlled experiments with computer languages. (Graver, 1977:6-9, 16)

The above discussion has briefly touched on some of the more significant factors which affect the development cost of software systems. Although full-scale development is usually the most costly phase of the software system life-cycle, the cost of maintenance is often significant. Brooks, for example, states that, "The total cost of maintaining a

widely used program is typically 40 percent or more of the cost of developing it." (Brooks, 1975:121)

The Cost of Maintenance. Graver points out that software maintenance has two components. These will be referred to as "error correction" and "system enhancement." It is Graver's position that only the error correction component of maintenance costs should be included in life-cycle cost estimates. This position is based on the argument that enhancement type modifications are concerned with product improvement rather than success or failure in meeting original product requirements. (Graver, 1977:6-33)

Putnam does not address this point explicitly. Some statements in his article, however, seem to imply that he intends to model only error correction maintenance costs. The continuously decreasing form of his resource utilization function reinforces this impression. (Putnam)

There is little question that the costs of error correction maintenance should be included in any life-cycle cost estimate. In addition, some enhancement decisions are clearly independent of the original system development. In other cases, however, the situation is not so clear-cut.

In the opinion of this author, some system enhancement maintenance activity is the direct result of management decisions made before or during development. As such, the costs of these enhancements should be included in system life-cycle cost estimates. Two examples of this include "bare-bones" systems and systems which must exist in a dynamic environment.

There are a number of reasons why management may decide to procure a "bare-bones" system (one which has only minimal capability). In some

cases, this may be done to insure early system availability. In others, it may not be clear what the mature configuration of a radically innovative system should be. In either of these cases, if current management decisions are based on the assumption that a bare-bones system will be enhanced, the costs of that enhancement should be included in the life-cycle cost estimate.

Another case where enhancement costs are relevant to life-cycle cost estimates involves systems which must meet changing user needs in a dynamic environment. One example of such a system would be a scientific system which, from its inception, is intended to continuously reflect the state-of-the-art. Another example would be a military command and control system. In many cases, the initial assessment of the wartime environment in which such systems will operate is already obsolete when the system becomes operational. In such cases, continuing enhancements to the software are necessary to ensure the continued viability of the weapon system in a wartime environment. Thus, life-cycle cost estimates should include the cost of such enhancements.

It is the opinion of this author that the maintenance portion of software life-cycle cost estimates should contain two components. The first is the cost of error correction, as recommended by Graver. The second is the cost of those nondiscretionary enhancements that are implicity in the assumptions inherent in the estimate. To date, empirical research on maintenance costs has concentrated almost exclusively on the cost of error correction, and even that has been minimal.

Graver's group investigated three questions concerning the cost of software maintenance, using data from two large, ground based command

and control systems. In the first case, they definitively established that the total number of problems reported for a system was related to the number of design changes and error corrections that had previously been incorporated into the system. (Graver, 1977:6-38) In fact, their results indicate that Brooks may have been optimistic when he estimated that each error correction has a 20 to 50 percent probability of introducing another error. (Brooks, 1975:122)

Graver also attempted to show that validation and verification (V&V) efforts were effective in reducing system errors. Unfortunately, this analysis was flawed, in that it was based on the number of errors found through V&V, rather than on the number of errors which escaped detection by V&V groups. (Graver, 1977:6-38, 40)

The last maintenance oriented question addressed by Graver's group dealt with the effect of programming language of the number of personnel required to maintain a particular system. They present convincing evidence that all other things being equal, it takes approximately four times as many programmers to maintain a MOL program, than a similar program written in a HOL. Thus, the claims that HOL's reduce overall maintenance costs seems to be justified. (Graver, 1977:6-40, 42)

An Introduction to PRICE S

PRICE S is a software development cost/schedule estimation model that is currently being developed by RCA PRICE Systems, Inc. It is based on the same parametric cost modeling methods that were used to develop their highly successful PRICE model for hardware cost/schedule estimation. A complete description of PRICE S is beyond the scope of this report. Instead, a brief description of the PRICE S inputs,

outputs, and operating environment must suffice. This discussion is based on a two day introductory course on PRICE S which was held at the RCA facility at Moorestown, New Jersey on 8 and 9 May 1977.

PRICE S inputs may be divided into three distinct categories: "hard" system parameters, "soft" system parameters, and "environmental" parameters. A full list of PRICE S input parameters is contained in Appendix A.

The environmental parameters describe the environment within which the system will be developed. They address such factors as escalation (price inflation) and rate of technological improvement. These parameters, permit users to vary the economic assumptions that they use in comparing historical data to the present. The present research does not address these parameters, except to use RCA's pre-set values without question. They will not be discussed further in this report.

The "hard" input parameters represent things which can be physically measured in the completed system. They include such items as total number of executable machine instructions and the proportion of the total instructions that represent each of seven different "application types." While they may or may not be well defined at the time a cost estimate is made, historical data can be used to generate firm measures of these quantities for any completed system.

The "soft" input parameters, on the other hand, are more subjective in nature, and cannot be measured directly from historical data. PRICE S uses three "soft" parameters that are relevant to the present research. Software Design Complexity (SDCPLX) is a measure of the "inherent difficulty" of the system development task. Theoretically, it relates the

difficulty of the task to the shop that is to accomplish the task. Engineering Complexity (ENCPLX), on the other hand, is a measure of how long the project "should" take. It seems to be based on the assumption that there is a "natural" project length that is characteristic of any system development project. Finally, Platform (PLTFM) is a measure of the reliability that will be required of the system. It seems to be related to the stringency of the system specifications.

The outputs of the PRICE S model (in normal mode) include cost and schedule estimates. The cost estimates are broken out in matrix form. The columns of this matrix are the three phases of the software development process: design, implementation, and test and integration. The rows are the five productive activities described above: systems engineering, programming, configuration management, documentation, and program management. The schedule estimates that are output include begin and end times (in months) for each of the three phases mentioned above. In addition to the normal mode outputs, PRICE S may be run "backwards" with historical cost and schedule information as input. In this mode, the model may be used to estimate values for SDCPLX and ENCPLX.

One unique characteristic of the cost estimation models developed by RCA PRICE Systems, Inc. is the concept of "boxing." Under this concept, each component of a system may be described and estimated individually as a separate "black box." The user may then estimate total system cost by using the output from each separate component estimation as input to a combined "system" estimate. The model automatically accounts for the additional "systems-level" work that would be required to design and implement the separate components as part of a coherent system instead of individually.

PRICE S has been implemented in a commercial time-sharing environment. Users contract with RCA PRICE Systems, inc. for the use of the program. This contract entitles them to access the program by "phoning" the time-sharing facility using a standard teletype terminal. In addition to the PRICE S model itself, the user also has access to a relatively sophisticated utility program which allows him to build, edit, and manipulate PRICE S input and output files.

III. METHODOLOGY

The Research Problem

Statement of the Problem. The problem addressed by this research is the need for a workable, accurate method of making preliminary software cost estimates at Aeronautical Systems Division (ASD).

Conversations with ASD cost analysts during the formative stages of the research have led this author to believe that preliminary budget estimates of software costs require the analyst to draw heavily on historical analogy. Even when parametric equations are used, analogy is required to estimate the size and difficulty of the system. The ability to draw analogies depends directly on the availability of descriptive, historical data.

Purpose. The purpose of the research is to investigate ways of gathering and using descriptive data for the purpose of making preliminary software cost estimates.

The orientation of the research toward the PRICE S model occurred primarily because several ASD cost analysts expressed an interest in the model. Further investigation revealed that Air Force Avionics Laboratory (AFAL) personnel were also interested in PRICE S. Through a fortunate coincidence, it was found that RCA PRICE Systems, Inc. was about to field test the program by allowing a group of prospective customers to use it on a trial basis at no cost. On the understanding that such participation in no way constituted an endorsement of PRICE S by the government, this author was allowed to participate in the test program.

Objectives. Two specific research objectives were chosen. The first objective was to develop a methodology for gathering descriptive data on software systems. The second was to use historical ASD software acquisition data to "calibrate" the PRICE S model.

The term "calibrate" can be used in two ways within a cost estimating context. ASD cost analysts speak of a cost analyst "calibrating himself" to a particular cost model. By this, they mean that the analyst has learned to choose input parameters for the model that result in accurate cost estimates. Thus, a "calibrated" cost analyst is able to correctly evaluate and adjust for subjective factors and model deficiencies in order to arrive at an accurate estimate. Although it is hoped that this research will contribute to the "calibration" of future ASD analysts, the specific objective of the research is to calibrate PRICE S in a different sense.

According to Webster's Seventh New Collegiate Dictionary, calibrate means, "to determine, rectify, or mark the gradations of ____." The "soft" PRICE S input parameters described above all use a numerical scale, but the individual values on these scales have no inherent meaning. It is only by associating known characteristics of real systems to particular values of these parameters that these values take on meaning. It is in this sense of "marking" individual values of the "soft" parameters by association with observed fact that the term calibration is used.

Scope and Limitations. The research applies to operational flight software systems procured by ASD System Program Offices (SPO's). Thus, only software systems that were actively involved in the flight and/or mission performance of aircraft in real-time were included. As a result,

any application of the results of this research to other types of software or other organizations should be done with caution.

Assumptions. The research is based on the following assumptions: Cost/Schedule information reported by contractors to SPO's is assumed to be a "reasonable estimate" of the true cost of the item involved. The validity of the PRICE S model is assumed. Thus, it is assumed that the model accurately reflects the effects of the various input parameters on software development cost and schedule.

Data Collection

The research objective of developing a methodology for gathering descriptive data on software was addressed through the data collection effort itself. This effort was designed to accomplish three objectives. First, it was used to screen prospective systems and eliminate those that were unsuitable for further study. Next, the data collection effort had to provide sufficient information to permit PRICE S model inputs to be generated. Finally, the effort was designed to acquire sufficient additional information to put the system in perspective. As it finally evolved, the data collection effort consisted of three parts: an initial interview, cost/schedule data collection, and statistical data collection.

The Initial Interview. A formal interview format, using both closed and open-ended questions was developed. This proved to be an effective screening device as well as a tool for gathering background information about individual systems. Because of the limited number of prospective systems, it was not feasible to formally validate the interview format. Instead, several knowledgable individuals were consulted

and their advice used in modifying the format. In addition, some questions were reworded when field experience showed them to be awkward or misleading. Because of the informal and free-flowing nature of the interviews, it is believed that these modifications did not affect the information gathered in any way. The interview format is contained in Appendix B.

Because only a small number of the prospective systems were able to pass the screening criteria, it is important to report what these criteria were. First, only operational flight software systems were considered. Of these, only those written in an assembly language were acceptable. This is because at present, PRICE S is not oriented toward HOL's. Next, the cost of the system development had to be easily derivable from the official reports provided to the SPO by the contractor. In general, this meant that software development was either a separate, reportable "line item" on the contract, or that the entire contract was for software system development. Finally, the system had to be at a stage of development where software development costs were known with a high degree of accuracy. The worst case that was accepted involved a system that was well into formal qualification testing with few apparent problems. Only two systems were able to meet these criteria.

Prospective systems were identified and the initial interviews were arranged through the Information Engineering Branch of the Directorate of Avionics Engineering at ASD (ASD/ENAI). This organization provides software engineering support to all ASD SPO's with ongoing avionics software development projects. The initial interview was normally held with the responsible software engineer in order to qualify the system

before other SPO personnel were approached. If the system looked promising, the responsible project officer was approached to collect the appropriate cost/schedule information.

Cost/Schedule Data Collection. Cost and schedule data proved to be the most difficult information to collect. Unfortunately, until very recently computer software was not generally recognized as a spearate components of avionics systems. As a result, for most of the systems investigated, software development costs were simply not available. These systems were dropped from further consideration. In those cases where software development costs were reported separately, several adjustments were made to attempt to make the reported costs more "realistic."

First, any separable nonsoftware costs were subtracted from the initial total. Of course, any identifiable costs that were directly allocable to software development were added to the total. In order to maintain consistency, it was decided to include "burden" (factory overhead) while excluding general and administrative (G&A) costs as well as fee or profit. The rationale for this decision was that the cost of work space and vital services is often a major component of burden, while G&A costs depend on the upper echelons of the organization rather than on the software development team itself. Profit, of course, is not normally included in cost estimates.

The major exception to the above policy was the treatment of "supporting contracts" such as Independent Verification and Validation (IV&V) contracts, or small research contracts that related directly to the software system. The total costs of such contracts, including

profit, were added to the software development cost on the theory that they were direct costs to the Air Force of acquiring the software. In accordance with common Air Force practice, no SPO operating costs or Air Force personnel costs were included.

Insofar as schedule was concerned, no attempt was made to separate the developments into phases. Each development was considered to have started when work began on the contract. This appeared to be a reasonable assumption, at least for the systems examined. The development end date was considered to be the date when either the Air Force formally accepted the software, or it was formally incorporated into a higher level system for the purpose of system testing at that higher level.

Statistical Data Collection. This was by far the most time consuming part of the data collection effort. Its purpose was to collect the data necessary to describe the software systems studied in terms of the "hard" PRICE S input parameters. It involved a statistical sampling of the individual modules of code that made up each system. Before the sample could be taken, however, several preliminary tasks had to be accomplished.

Once the initial interview and the cost/schedule data had been collected, the researcher normally spent several hours becoming familiar with the system documentation. With the assistance of the software engineer who was most familiar with the system being studied, the researcher was always able to find the information required to collect the data sample. This inlcuded: a complete list of all the modules in the system, a method of tracing the functional hierarchy of the modules,

a system diagram that related the modules to "major" functional tasks, and a complete copy of the system assembly language code. When the researcher was satisfied with his ability to work with the available documentation, he could address the next step.

The last task other than the sampling itself was to establish a means of associating each module chosen with a single PRICE S "applications type." This was done with the help of the software engineer. Using descriptive terms similar to those in Table III, each "application type" was described in terms of the additional considerations or constraints which acted to make the programmer's job more difficult than it would otherwise be. The concepts involved were discussed until both parties agreed that a common understanding had been reached. The researcher then asked the engineer if it would be reasonable to assign a single "application type" to each functional area on the system diagram mentioned above. In no case did the researcher encounter any problem or resistance to the allocation of "application types" to modules on a simple, functional basis. Once this was accomplished, statistical sampling could begin.

Originally, the researcher planned to look at every module in every system studied. This quickly proved to be impractical. As an alternative, a sequential sampling technique was developed which resulted in a substantial reduction in the time necessary to evaluate a particular system.

The sequential sampling technique was based on the "weighted instruction count" (weight) of each module. In equation form:

$$w_i = d_j n_{ij}$$

TABLE III

PRICE S Application Types

Application Type	Weight	Identifying Characteristics
System	10.95	Heavy hardware interface. Many interactions. High reliability and strict timing requirements. Typified by operating systems.
Interactive	10.95	Interfaces with human operators. Human engineering considerations and error protection very important.
Real Time Command and Control	8.46	Real time communications under tight timing constraints. Queueing not practicable. Strict protocol requirements. Heavy hardware interface.
On-Line Communications	6.17	Real time communications with queueing allowed. Timing restrictions not as restrictive as with REAL TIME COMMAND AND CONTROL.
Data Storage and Retrieval	4.10	Secondary storage handling. Data blocking and deblocking. "Hashing" techniques. Hardware oriented.
String Manipulation	2.31	Routine applications with no overriding constraints. Not oriented toward mathematics. Typified by language compilers.
Mathematical Applications	0.87	Routine mathematical applications with no overriding constraints.

where: w_i = the "weight" of the i^{th} module.

d_j = the "density," or weighting factor associated with application type j.

n_{ij} = the number of instructions in module i which happens to be of application type j.

The weight of the entire system is then simply the sum of the weights of the individual modules. System weight is the basis from which cost is calculated in the PRICE S model (Freiman, 1977).

Appendix C contains a detailed derivation of the sequential sampling equation used in the research. Briefly, an initial, random sample of 50 to 100 modules was chosen without replacement. The number of executable machine instructions, application type, and hierarchical level were then recorded for each module chosen. The number of modules required to obtain a 95 percent confidence level that the mean module weight was within plus or minus ten percent of the sample mean was then calculated using:

$$n = \frac{N}{(1 + (\bar{w}^2/384)(N-1)/s_w^2)}$$

where: n = the desired sample size.

N = the total number of modules in the system.

$\bar{w}$ = the mean weight of the latest sample.

s_w^2 = the sample variance of w (module weight).

Sampling without replacement was then continued until the cumulative sample size reached n. At this point, a new sample mean and variance were calculated and the process was repeated until the new value of n

was less than or equal to the existing sample size. The weighting factors used for each application type were those being used by RCA PRICE Systems, Inc. on 9 May, 1977. They are listed in Table III.

Methods of Analysis

A number of analytical methods were used to accomplish the research objectives described above. Subjective evaluation was used to analyze the data collection methodology. Statistical analysis techniques were used to calculate PRICE S parameters and analyze possible improvements to the data collection methodology. In addition, experimental design and sensitivity analysis techniques were used to calibrate the PRICE S model. Each of these methods is discussed briefly below.

Subjective Evaluation. The analysis of the data collection methodology was, of necessity, primarily subjective in nature. It was oriented toward two, related goals: analyzing data collection problems, and evaluating the usefullness of the data actually collected.

Statistical Analysis. With the exception of the sequential sampling technique described above, the statistical analysis methods used in the research were neither extensive nor complicated. Point estimation techniques were used to calculate PRICE S input parameters and other descriptive statistics. In addition, "t-tests" were used to test some hypotheses concerning possible improvements to the statistical data collection methodology. Freund's Mathematical Statistics contains an excellent discussion of both of these techniques. (Freund, 1971)

Although it is not a formal statistical analysis technique, it should be noted that frequency distribution plots were used to graphically illustrate the probability distribution of "number of instructions per

module" within the software systems studied. To generate this type of plot, the possible values which a particular random variable may assume are divided into an ordered set of equal sized intervals, called cells. A random sample of that variable is then selected. The proportion of times that the variable assumes a value within each cell is then plotted against the ordered set of cells. The resulting plot approximates the shape of the probability density function for the random variable in question.

PRICE S Methodology. The data collected on each system was used to calculate the initial inputs for the PRICE S model. Since all systems studied fell under the general category of "military avionics," the "soft" input parameter PLTFM, which measures the stringency of the development specifications, was set to 1.8. This is the value recommended by RCA PRICE Systems, Inc. for "MIL-Spec Avionics" systems. (RCA PRICE Systems, 1977:27) Using these values, a preliminary PRICE S run was made to calculate initial values for the "soft" parameters SDCPLX and ENCPLX, which relate the difficulty of the development task to "the shop doing the work," and "the normal time required for its accomplishment," respectively. (RCA PRICE Systems, 1977:12, 20)

Once the initial run had been made, the researcher made a series of runs to evaluate the results of "adjustments" to the initial input parameters. Even with the ground rule that no parameter which had been objectively measured could be altered, this was a very subjective process. The intent of the adjustments was to find a set of input parameters which both included the objectively measured data and satisfied the researcher than it adequately reflected his subjective evaluation of the development

effort. The result of the adjustment runs was a set of "baseline" PRICE S parameters, which the researcher believed best reflected the system development being studied.

In order to provide for comparability between systems, the baseline parameters were then input to a "baseline" PRICE S run. For this run, the project cost and schedule were estimated in constant 1977 dollars, with the project start date (ENDSS) adjusted to 1 February 1977. (The February date is convenient for PRICE S users because of peculiarities in the computer program.) This baseline run was the starting point for a series of twenty PRICE S runs that were designed to provide data for a sensitivity analysis of the model.

Six parameters were selected for a formal analysis of the sensitivity of the PRICE S model to input parameter changes. Three of these parameters were believed to have a realtively simple, straightforward effect on the cost and schedule estimates output by the model. These parameters were: "the total number of instructions" (TNINST), "the instruction density" (IDENS), and "fraction of available capacity utilized" (UTIL). To analyze the effects of these variables upon estimated cost and schedule duration, each of these parameters was varied separately, in small increments about its baseline value while the other PRICE S inputs were held constant. Since the baseline run acted as the center point for each variable, four PRICE S runs were sufficient to establish five data points for each of the three parameters examined in this way.

The other three parameters which were evaluated were the three "soft" parameters described above: SDCPLX, ENCPLX, and PLTFM. Because the effects of variations in these parameters was not necessarily either

simple or straightforward, it was decided to use a more sophisticated experimental design for this part of the analysis. In order to provide sufficient data for an analysis of the interactions between these parameters without unnecessarily increasing the number of PRICE S runs required, it was decided to use a full factorial experimental design with two levels for each of the three variables being analyzed. Thus, eight PRICE S runs were required to reflect all possible combinations of these input variables. Since the PRICE S model is deterministic in nature, in that the same inputs always produce the same outputs, it was not necessary to either randomize or replicate these runs.

Sensitivity Analysis Techniques. Two mathematical techniques were used to analyze the results of the sensitivity analysis runs. These are linear regression and analysis of variance (ANOVA). As used in this study, linear regression is a widely accepted method of fitting linear equations to numerical data, and ANOVA Is a method of analyzing the degree of interaction between independent variables in their effect on dependent variables. Since the data being analyzed was not the result cf random (or pseudo-random) processes, these techniques were not used as tools of statistical analysis. The interested reader is referred to Draper and Smith for an excellent treatment of regression, as well as an interesting approach to ANOVA. (Draper and Smith, 1966) Freund provides a more traditional discussion of ANOVA. (Freund, 1971)

System Descriptions

The final phase of the research was the development of a format for recording historical data on software systems. This format is intended to provide a medium for recording the essential facts necessary to

describe a historical software system in PRICE S terms. This format was developed with a number of guidelines and restrictions in mind. First, it was decided to restrict the format to no more than one page in length. At the same time, it was desired that the format be as simple and self explanatory as possible. In addition, while the prime emphasis of the format was to record factual data, sufficient information to communicate the purpose and scope of the system should be provided. Finally, there had to be a way of recording any "special case" information that would be of interest to another cost analyst. The characteristics of the two systems studied were recorded on the new format. This information is presented in Appendix D.

IV. ANALYSIS AND RESULTS

Overview of the Systems Studied

All in all, ten software systems were considered for inclusion in this study. These included software developed for the following weapons systems managed by ASD:

B-1 (two systems considered)

AMST

EF111

ALQ131

PAVE TAC

F-16

ALS

Wild Weasel

F-15

Although all the project personnel who were contacted were most cooperative, only two of the ten systems passed the screening criteria described above in Chapter III (The Initial Interview). These systems will be referred to in this report as System A and System E. Each is described briefly below.

System A. This software system is part of a sophisticated electronic jamming system. The software analyzes and identifies radar signals and allocates jamming resources in order to counter threats. Although the software is resident on two separate computers, it was developed by a single team under unified management. In general, the software on one machine does the actual jamming tasks, while the software on the other machine controls the interface with the human operator. The entire system

is resident on a fighter type aircraft. Under combat conditions, failure of this software system could significantly degrade the effectiveness of the jamming and thus endanger the aircrew.

System A was developed concurrently with the rest of the jamming system. At the time that the data on this system was collected, the software development was not yet complete. The system was, however, well into formal qualification testing with few apparent problems. For this reason, budget estimates were accepted for the uncompleted portion of the development effort. The cost account which contained most of the costs for this system included the direct costs of the software development including payroll costs for the manager of the programming effort. Some relevant costs, however, were in other accounts and could not be recovered. These included the cost of systems engineering, which was recorded in a separate account that included all systems engineering work for the jamming system. In addition, a small protion of the relevant documentation costs were recorded in a separate "data" account. The cost of a relatively small IV&V effort was also considered to be relevant to the software development effort.

So far as could be determined from discussions with the ASD project manager and the software engineer, the development team was well-qualified, but not exceptionally so. It was felt that the schedule was neither unusually accellerated nor extended. The software specification requirements were described as "leaning toward the lax side." There seemed to be no serious problems with hardware capacity, especially since the system was designed to minimize the impact of hardware timing problems.

System E. This software system is unusual for an operational flight system, since it does not leave the ground. It does exercise control over an aircraft in flight, in real-time, however, so it must be considered to be in the same class as other operational flight software systems. Its purpose is to navigate and control a guided munition along a given trajectory. The software is resident in a ground-based computer which communicates with the munition and with its parent aircraft through a real-time data link. It was developed as part of an advanced development effort to build a prototype weapon system. Failure of this system could result in loss of control of high explosive munitions in flight, this could endanger human life.

System E was developed concurrently with the rest of the prototype weapon system. The software developer was not responsible for most of the hardware development, which was under a separate contract. Several other tasks, not related to the software development, were included in the software development contract. The costs of these tasks were reported separately, however, and could be "backed out" of the total contract costs. The ASD project manager who was responsible for monitoring the software development effort reviewed the researcher's cost calculations. He indicated that although some nonrelevant costs were probably still contained in the total, they were probably not large enough to be considered significant.

The System E software development team was considered to be exceptionally well qualified for the task by the ASD software engineer who was most familiar with the project. In addition, this individual

indicated that there was considerable pressure to finish the development more quickly than "normal." The specifications were described as being lax, but did not seem to be inordinately so. Computer memory capacity was perceived as a problem, but hardware timing was not. As with System A, this system was designed to minimize the impact of hardware timing problems.

Evaluation of the Data Collection Methodology

The Intital Interview. The interview format contained in Appendix B served several useful purposes. It provided the researcher with an overview of the software system. It also served as a filter, allowing the researcher to determine within an hour or less whether or not the system in question was a suitable subject for further study. In addition, it served as a simple medium for collecting some of the descriptive information, both objective and subjective, that should be included in a historical software data base. In particular, the questions concerning the operating environment of the software system, the computer on which the system would operate, and the functions performed by the system are essential to the description of any software system. A few points concerning the initial interview are worthy of special note.

The list of "major sub-functions performed" shows promise of becoming an excellent tool for assessing how comparable two systems really are. The intent of the question was to identify the component functions which the system performed in achieving its ultimate purpose. The software engineers contacted by this researcher were always able to provide complete, concise answers to this question with very little effort. Given this information, it should be possible to compare the list of functions

performed for two purportedly similar systems. Such a comparison should quickly reveal whether or not the similarity is only superficial. It should also reveal the major functional areas of difference between two similar systems. Of course, other considerations such as the method of implementation used and the accuracy required must be considered before the similarity between the systems can be established in detail.

Not all the questions on the interview format worked as well as anticipated, however. In retrospect, it is now apparent that the question which asked for an estimate of the percentage of system capacity actually utilized did not work as anticipated. Such a determination should be based on objective test data, or measurements made by the researcher. This question is useful for determining whether or not system capacity problems affected the development effort. Given an indication that such problems existed, however, the researcher should seek objective data concerning the extent of the problem.

There are several ways that the researcher may seek objective data concerning the fraction of system capacity utilized. First, if the project was of any size and the problem was serious, there should be some correspondence, formal reports, or other documentation of the extent of the problem. If not, the researcher may attempt to estimate the extent of the problem himself. For simple memory capacity problems, where no secondary storage is involved, it may be possible to determine the fraction of system capacity utilized by analyzing a system "load map." Also, a simple estimate of processing capacity utilized may often be calculated from formal test reports. However, if system capacity usage cannot be defined in a simple, straightforward manner, the researcher should not

attempt to estimate the fraction of capacity utilized without assistance from knowledgeable systems engineers.

Cost/Scehdule Data Collection. The collection of accurate cost and schedule data for avionics software development projects poses some unique problems even if the trend toward more detailed reporting of system development costs continues. Based on discussions with the ASD personnel responsible for monitoring the development of ten separate avionics software systems, it is apparent that such software is seldom procured by itself. Rather, avionics software is normally developed as an integral part of a larger system which is made up of both hardware and software components.

Under such conditions, two very basic problems arise for the researcher who is interested in isolating software cost/schedule data.

First, it will normally be advantageous, perhaps even essential, for the system developer to organize some functions that contribute to software development at the overall project level. This is especially true of systems engineering; to a lesser extent it applies to configuration management, documentation, and program management as well. Normally, only the direct cost of the programming function itself will be directly allocable to software development. This problem is compounded for ASD because each contractor must be allowed the latitude to organize his development team as he sees fit. Thus, the existence of a separate, project-wide systems engineering cost account will mean different things for different projects. This, combined with the fact that systems engineering work is, by its very nature directly allocable to a specific system component such as software, makes the "true cost of software

development" quite difficult to define, much less measure. In the opinion of this researcher, this problem is endemic to the way that systems are procured at ASD. As a result, any attempt to force all costs that are relevant to software development into a discrete set of cost accounts would be futile, if not actually counterproductive. Instead, it would be better for future researchers to agree on a consistent convention for allocating systems engineering costs to software development. Such an allocation could, for example, be made on the basis of the "direct engineering man-hours" associated with the design of the various system components.

Another, related problem concerns the length of the software development effort. At some point in time, avionics software ceases to be a separate entity and is incorporated into the system of which it is a part. Ordinarily, this occurs well before the system development itself is complete, since the entire system must be tested. If the software development team was disbanded at this point, and no more costs were incurred for software, this would create no problems. This is seldom the case, however. The software development team normally transitions naturally from the development phase of the software system life-cycle to the maintenance phase while the overall project is still in development. Thus, software related costs continue to be incurred even after the software development is complete.

The solution to this problem is not nearly as difficult as the cost allocation problem. The formal certification that the software is ready for testing at the next higher system level could easily be made a congractural milestone. The date on which this milestone occurs should be considered to be the end of the software development effort for historical

data collection purposes. Any software related costs occuring after this date should be allocated to software maintenance. In the case of larger systems, where it is advantageous to control software on the basis of a unit smaller than the entire system, this transition could be considered to occur on a unit-by-unit basis, rather than as a single, system-wide transition.

Statistical Data Collection. The sequential sampling technique that was used to measure the proportion of instructions that were of each "application type" served its purpose of significantly lowering the number of modules that had to be examined. The distribution of "number of instructions per module" that was encountered, however, caused this technique to be less efficient than was originally anticipated. Figures 2 and 3 show the actual distribution of "instructions per module" that was encountered for the two systems examined. These figures show clearly that there was a heavy preponderance of small modules. In both cases, the most frequently occurring number of instructions per module was less than twenty. Application of the "weighting factor" (due to PRICE S application types) to the number of instructions per module did not appreciably change the shape of these distributions, as is shown by Figures 4 and 5.

Table IV summarizes the calculation of the number of modules to be sampled for the two systems examined. In both cases, the number of modules required was significantly more than one half of the total number of modules in the system. It must be noted that in the case of System E, two extremely large "real-time control" type modules, which together contributed almost 14 percent of the total "weight" of the system were eliminated from the calculations. This was done for two reasons.

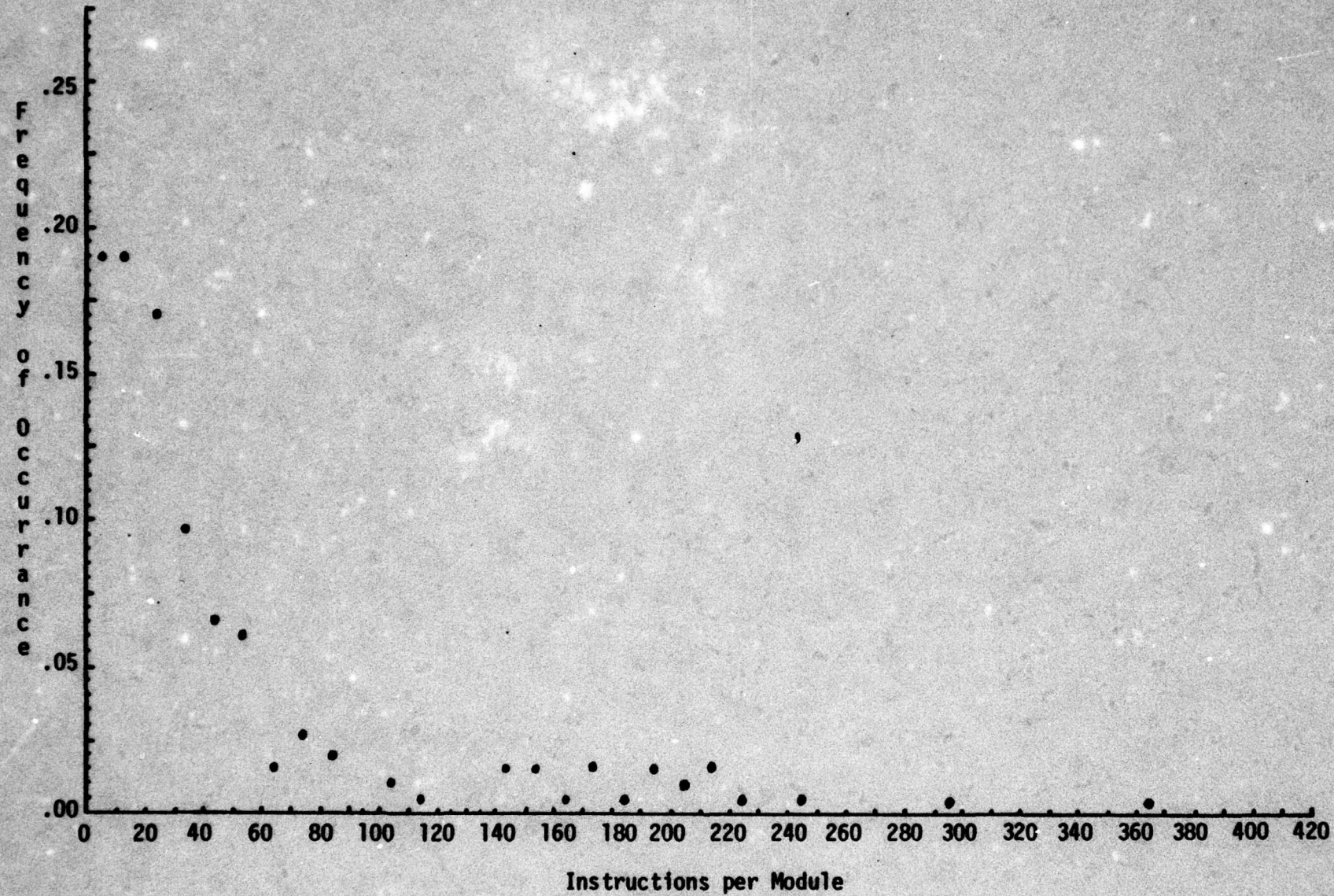


Figure 2. Frequency Distribution for Instructions per Module - System A

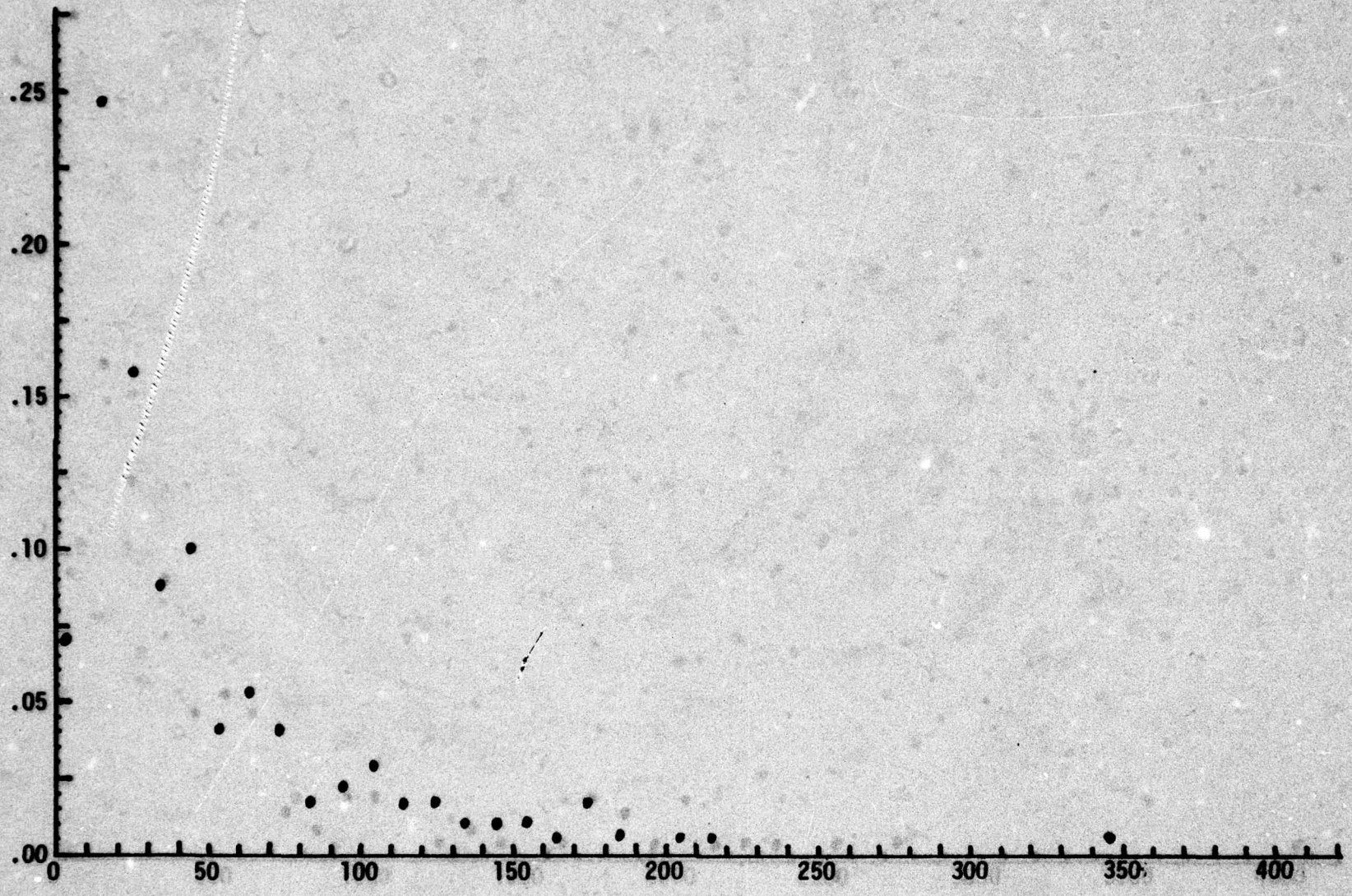


Figure 3. Frequency Distribution for Instructions per Module - System E

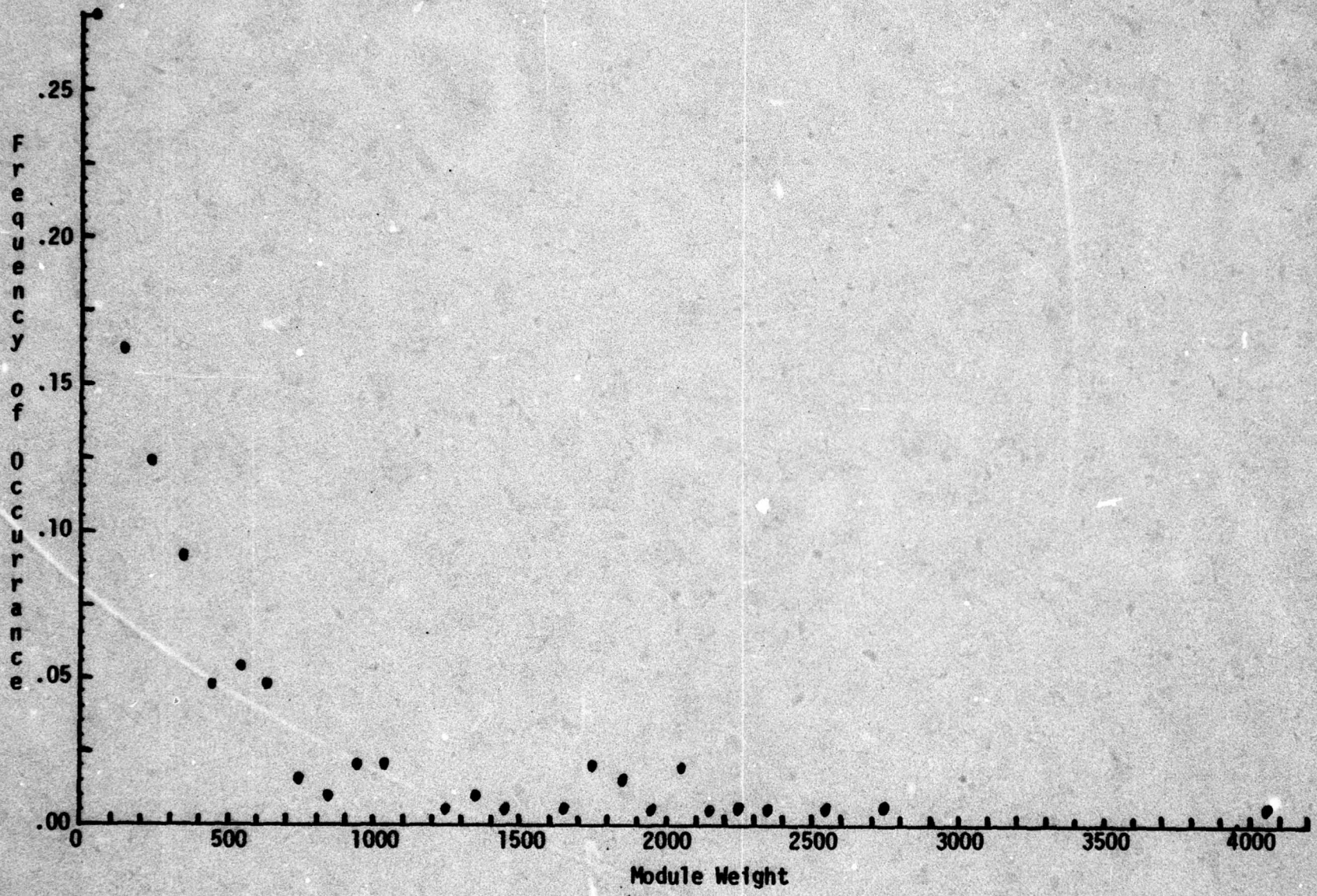


Figure 4. Frequency Distribution for Module Weight - System A

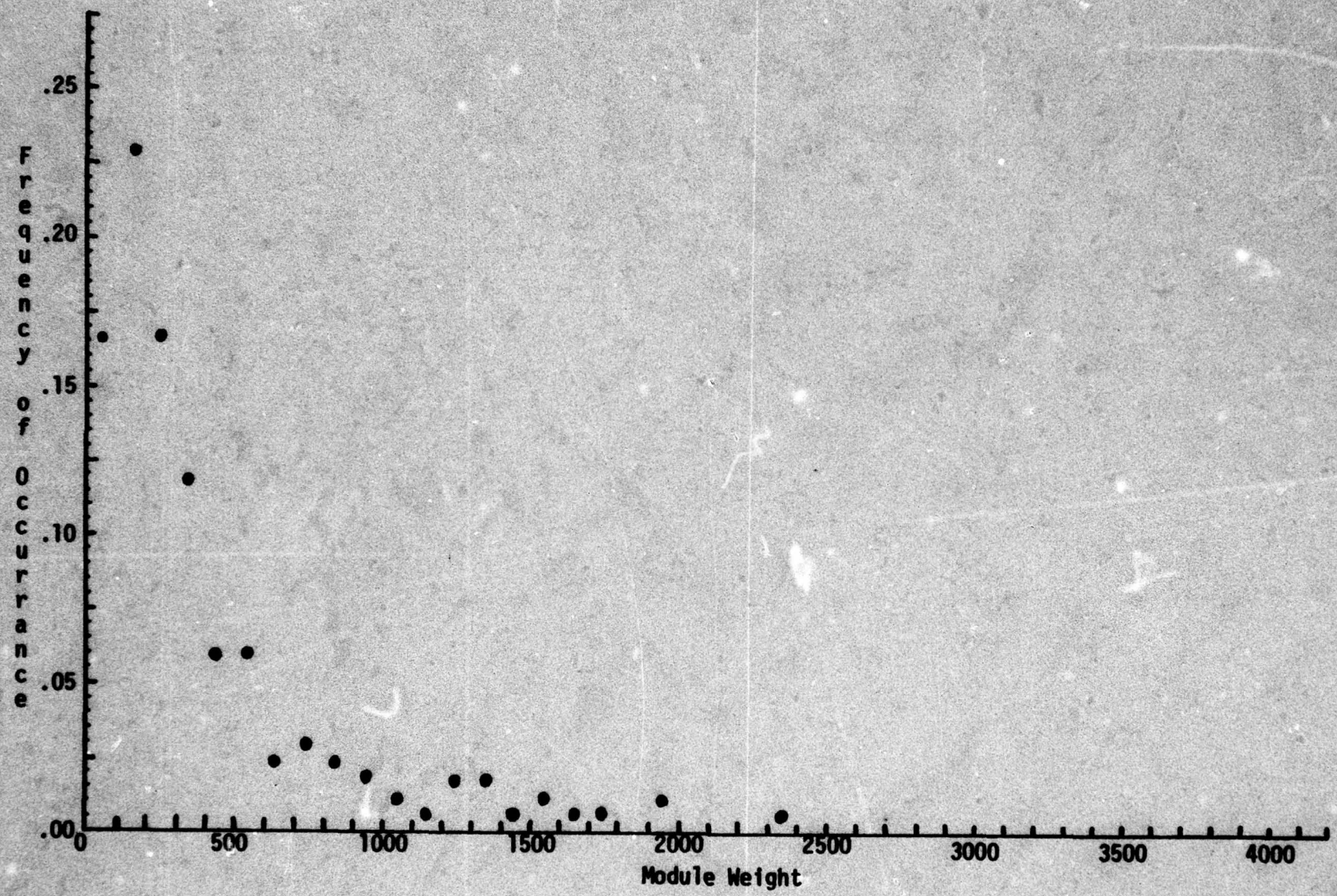


Figure 5. Frequency Distribution for Module Weight - System E

TABLE IV

Summary of Sample Size Calculations

Variable	Explanation	Values System A	Values System E*
N	Total modules in system	307	295
$\bar{w}$	Mean "weight" per module	491.4	415.6
s_w	Standard deviation of weight per module	648.3	433.4
p	Fraction of total modules required for the sample	0.686	0.587
n	Number of modules required in sample	211	173

*two modules were deleted from the System E calculations.

First, there was evidence that these two modules were unique, and that they may have been incorporated into the system virtually unchanged from previous work done by the developer. In addition, their combined effect on the total number of modules required for the sample was significant. Eliminating these two modules reduced the proportion of modules required for the sample by 17 percent (from 75% to 58%), a change of about 50 modules. Of course the effects of this decision on the other calculations regarding System E were evaluated. In no case was a significant change to any of the calculated system parameters discovered.

This researcher has discussed the observed distribution of "number of instructions per module" with several individuals who are familiar with avionics software systems. As a result of these discussions, it seems reasonable to assume that the distributions observed are typical of this type of system. If so, and if the experience of this researcher may be considered to be typical, future researchers may expect to devote

approximately two man-weeks to the measurement of a single avionics software system containing approximately 300 individual modules. Approximately one half of this time will be spent examining individual modules and physically counting instructions. The extension of this technique to larger software systems would involve proportionally more work.

In an effort to reduce the effort that future researchers would have to devote to statistical data collection efforts of this type, the researcher investigated the possibility that the need to count individual machine instructions could be avoided. The most obvious way to do this would be to calculate the "mix" of application types for a given system using the number of modules of each type rather than the number of instructions in these modules. The data from both Systems A and E were used to perform these calculations. The results, shown in Table V, indicate that the "mix" calculated using the two methods is quite different. Thus, a simple elimination of the requirement to count instructions did not seem to be warranted.

One further effort was made to eliminate the requirement for future researchers to count individual instructions. It was reasoned that if modules of each "application type" could be shown to contain the same mean number of instructions, independent of the software system involved, then a simple adjustment to the weights for each application type could be made. It was thus hypothesized that: the mean number of instructions per module for each application is equal in Systems A and E. This hypothesis was tested using the data on the "interactive" modules from the two systems, and a two-tailed t-test for the equality of means. The results of this test, shown in Table VI, indicate that the hypothesis

TABLE V

Comparison of Alternative Methods of Computing "Mix"

SYSTEM A

Application Type	Weighting Factor	Fraction of Modules Sampled	Fraction of Instructions Sampled
System	10.95	.146	.162
Interactive	10.95	.421	.321
Real Time	8.46	.151	.130
On Line	6.17	.140	.153
Data	4.10	.027	.051
String	2.31	.114	.182
Math	.87	.000	.000

IDENS CALCULATED USING Module Data	Instruction Data
8.71	7.95

SYSTEM E

Application Type	Weighting Factor	Fraction of Modules Sampled	Fraction of Instructions Sampled
System	10.95	.135	.112
Interactive	10.95	.435	.265
Real Time	8.46	.276	.456
On Line	6.17	.059	.034
Data	4.10	.000	.000
String	2.31	.006	.011
Math	.87	.088	.123

IDENS CALCULATED USING Module Data	Instruction Data
9.06	8.36

TABLE VI

Comparison of "Interactive" Modules From Two Systems

Variable	Explanation	DATA System A	DATA System E
n	Number of modules in data sample	78	74
$\bar{x}$	Mean number of instructions per module	46.923	34.946
s	Sample variance (x)	3096.59	1710.98
Hypothesis:	$\bar{x}_A - \bar{x}_E$		
Alternate:	$\bar{x}_A - \bar{x}_E \neq 0.$		
Equation:	$z = \frac{\bar{x}_Z \quad \bar{x}_E}{[(s_A^2/n_A) + (s_E^2/n_E)]^{\frac{1}{2}}}$		(Freund, 1971:319)
Therefore:	$z = 1.51$		
	$\hat{\alpha} = .131$ (two-tailed test)		
Conclusions:	The hypothesis cannot be formally rejected. The alternate hypothesis is more likely to be true than the null hypothesis.		

should probably be rejected, although the confidence level is only 0.87. Thus, this researcher was not able to identify a method of obtaining statistical data equivalent to that collected in this study without counting instructions.

PRICE S. Results

The results of the PRICE S runs fall logically into two groups. First, the initial runs, and the adjustments necessary to establish a set of

baseline PRICE S parameters will be discussed for each of the two systems. Finally, the results of the sensitivity analysis will be addressed.

System A Adjustments. The initial PRICE S run for System A yielded a value of SDCPLX that was less than two. This is indicative of a system which is extremely easy for the shop doing the work. This was not in consonance with the information gathered in the initial interview. Estimating the total cost for the system using the more "reasonable" value of 3.5 for SDCPLX yielded a total cost that significantly exceeded the measured cost of the software. To resolve this discrepancy, the cost of the systems engineering function was deleted from the PRICE S estimate. (The program provides a simple means of accomplishing this.) The decision to make this adjustment was based on the fact that systems engineering costs were recorded in a separate, project-wide account and were not included in the measured software development cost. Once this adjustment had been made, the value of SDCPLX which the model produced was raised to a value of 3.3. The researcher interprets this value as indicating a competent, but not exceptional work group working on a project which is well-defined and similar to previous work. The baseline PRICE S parameters for System A are listed in Table VII.

System E Adjustments. The baseline parameters for System E were established with much less trouble than was required for System A. This does not indicate, however, that these parameters are more creditable than those of System A. To the contrary, the ease with which the baseline was established reflects the researcher's inability to establish firm "acceptability limits" for the values of three key parameters: SDCPLX, ENCPLX, and UTIL.

TABLE VII

Baseline PRICE S Parameters

	System A	System E
Descriptive Parameters		
TNINST	19,000	16,000
IDENS	7.965	8.365
SDCPLX	3.3	3.3
FUNCT	307	297
FCONST	4.44	2.848
LEVEL	2.6	4.0
"Mix" Parameters		
MATH	.00	.12
STRING	.18	.01
SYSTEM	.16	.11
DATA	.05	.00
ON-LINE	.16	.03
REAL-TIME	.13	.46
INTERACTIVE	.32	.27
Other Parameters		
ENCPLX	2.26	1.9*
PLTFM	1.8	1.8
UTIL	0.6	0.8*

*These parameters were chosen by the researcher to reflect subjective considerations as well as objective measurements.

As stated above, the initial interview revealed that the development team was considered to be exceptionally well qualified for their task. In addition, there was considerable pressure on the team to shorten the software development scheduled. Finally, hardware capacity limitations were perceived to be a problem by the individuals familiar with the project. No objective measure was made of the actual fraction of the system capacity tuilized. Since only two PRICE S model outputs (total system cost and schedule duration) were considered in this study, uncertainty in three input parameters meant that the model solution was underdetermined. Thus, any number of "acceptable" combinations of values of SDCPLX, ENCPLX, and UTIL could be used to generate the desired values for the cost and schedule outputs. Thus, the actual baseline values chosen (Table VII) must be considered to be only typical of an infinite set of equally acceptable values.

Sensitivity Analysis. The purpose of the sensitivity analysis was twofold. First, it was desired to determine the reaction of the PRICE S model to small changes in the various input parameters in the neighborhood of the system baseline parameters. Second, it was desired to see if there was any significant interaction between the "soft" input parameters in their effect on the model cost and schedule outputs. The data necessary to achieve both of these purposes was generated from the 20 "sensitivity analysis" PRICE S runs described in Chapter III.

Since the PRICE S model is based on parametric equations, it was expected that the surfaces described by the sensitivity analysis runs would appear to be smooth and continuous. Such was indeed the case. Figures 6 through 17 summarize the results of the runs which allowed the parameters TNINST, IDENS and UTIL to vary independently. From these

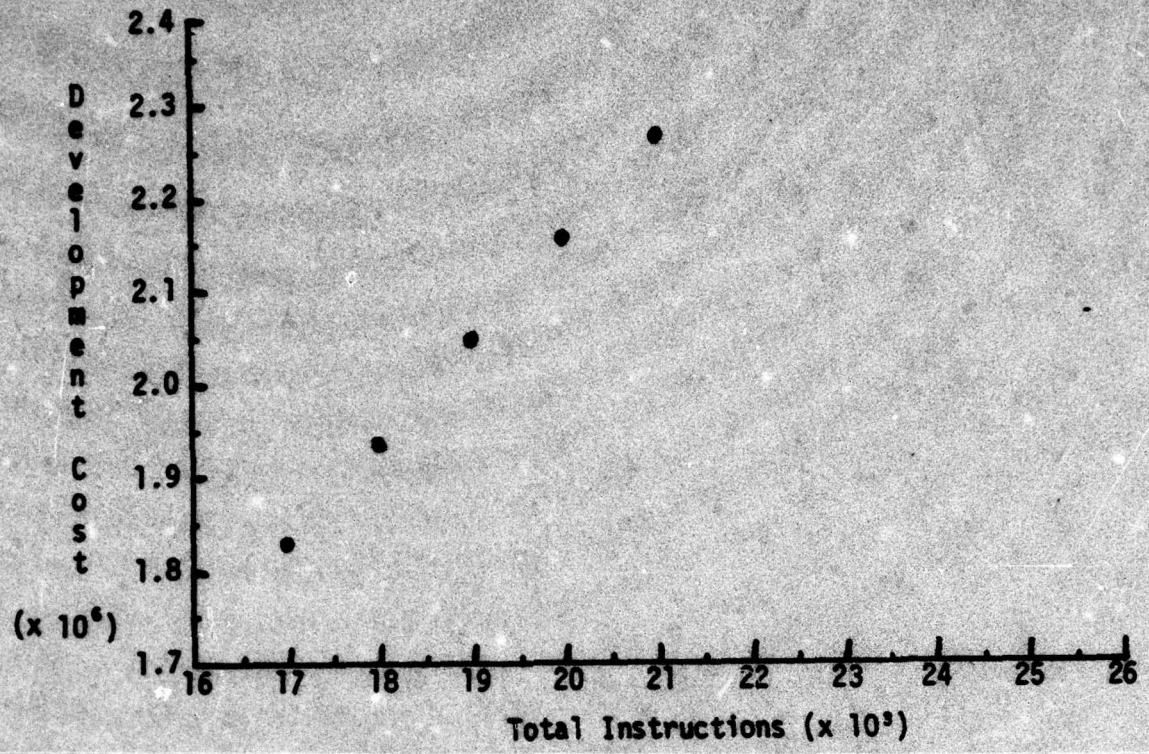


Figure 6. TNINST vs. COST - System A

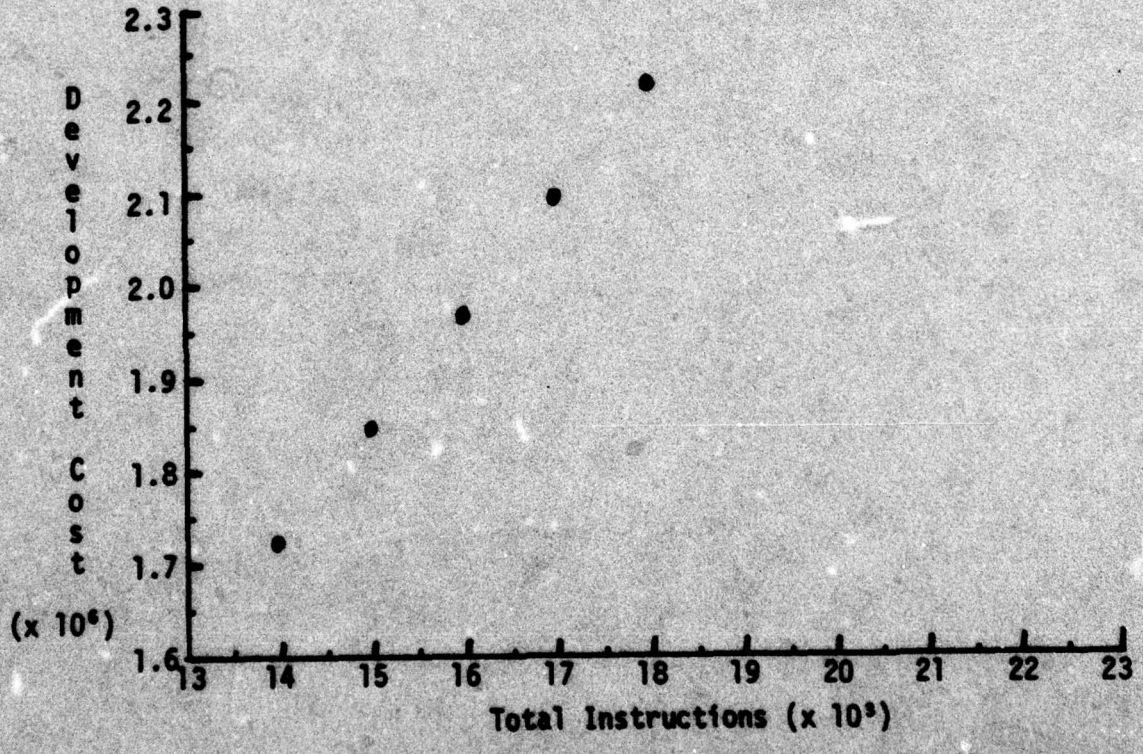


Figure 7. TNINST vs. COST - System E

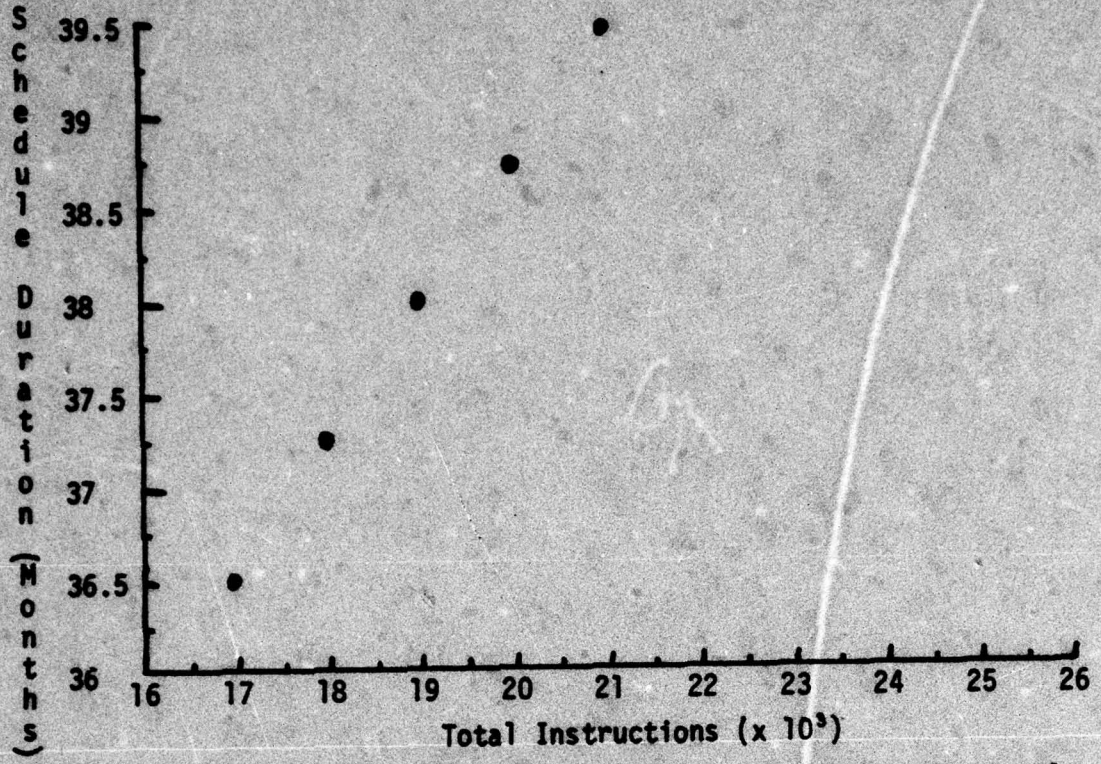


Figure 8. Total Instructions vs. Schedule - System A

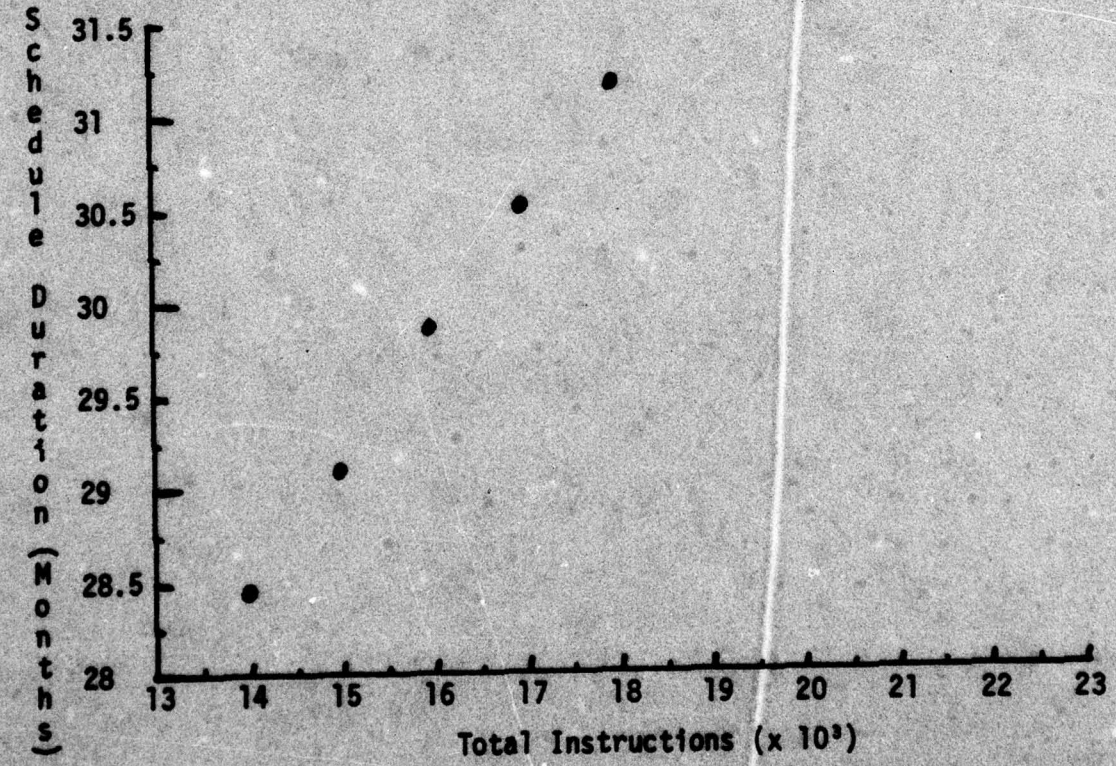


Figure 9. Total Instructions vs. Schedule - System E

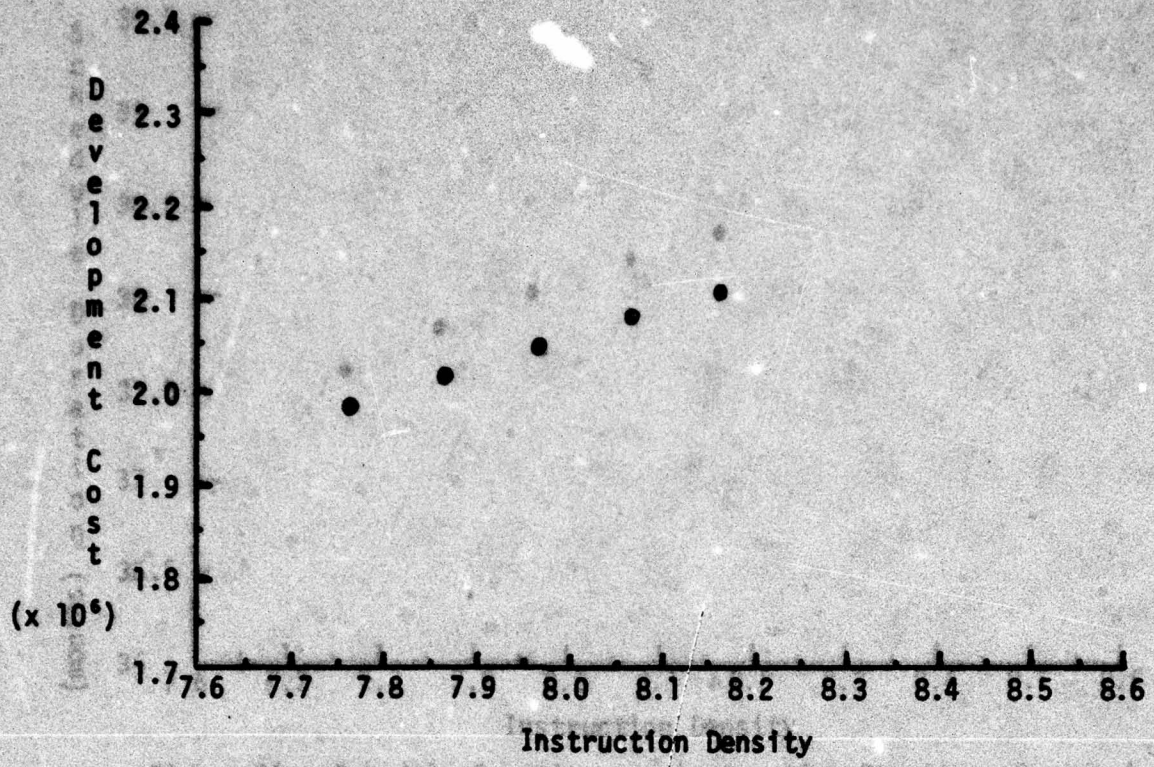


Figure 10. Development Cost vs. Instruction Density - System A

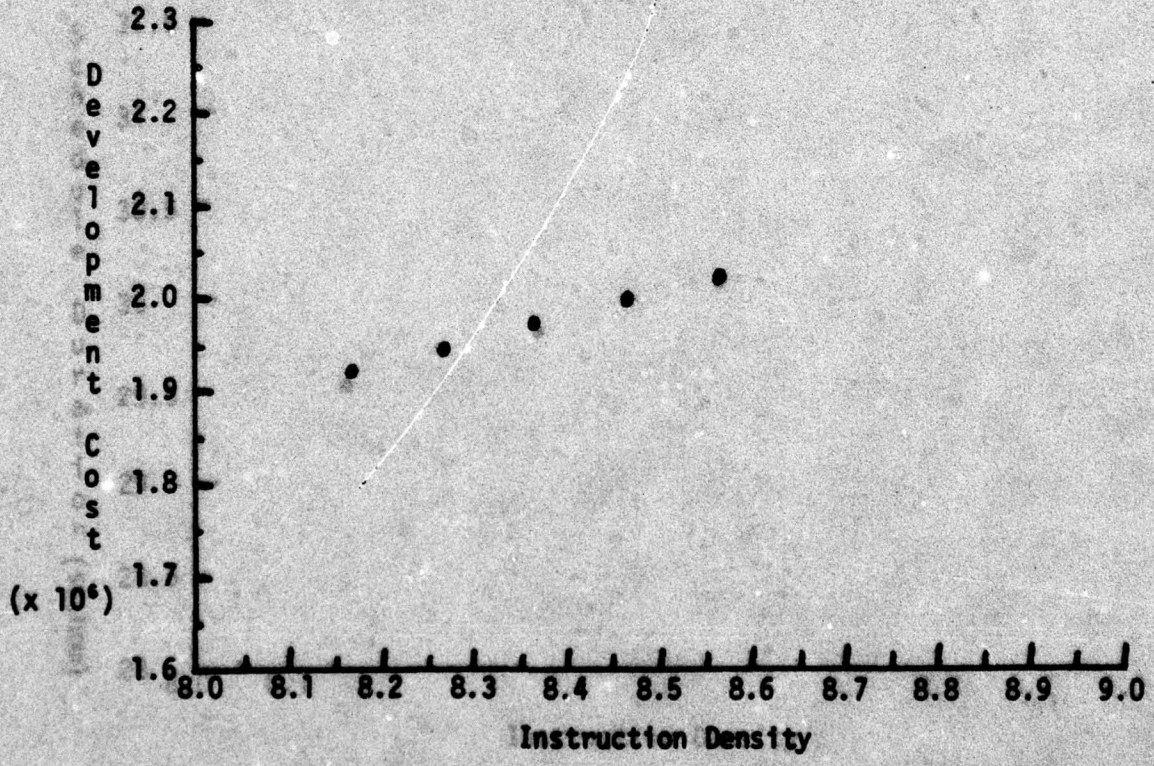


Figure 11. Development Cost vs. Instruction Density - System E

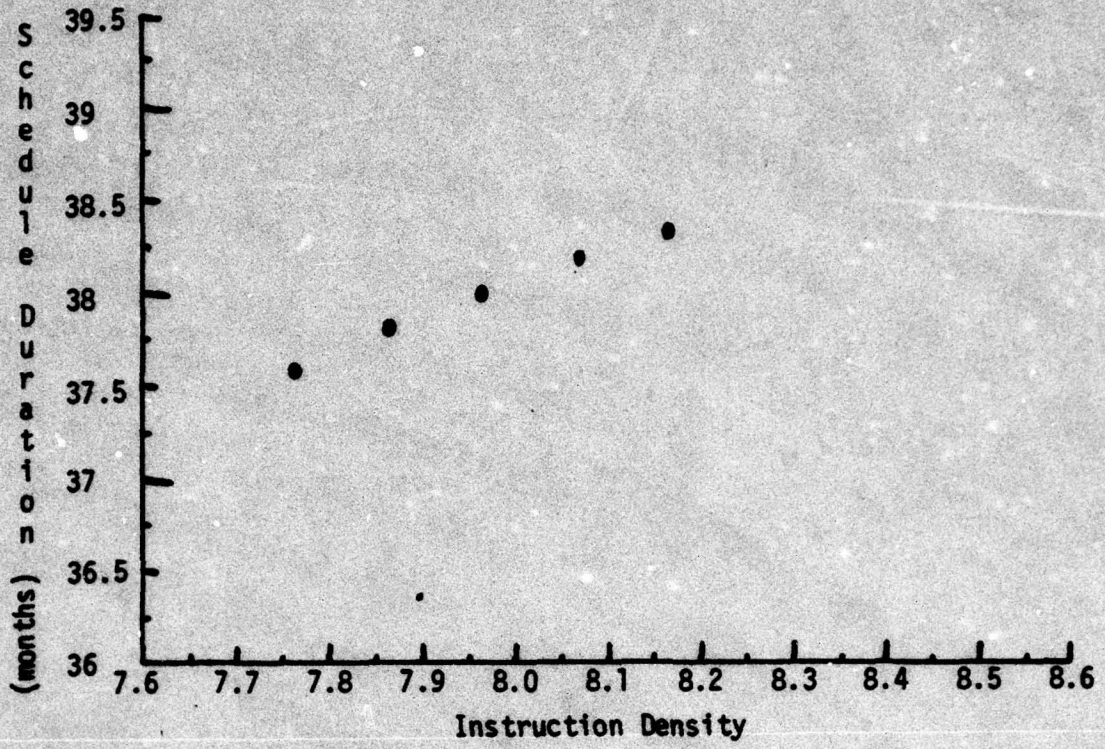


Figure 12. Schedule Duration vs. Instruction Density - System A

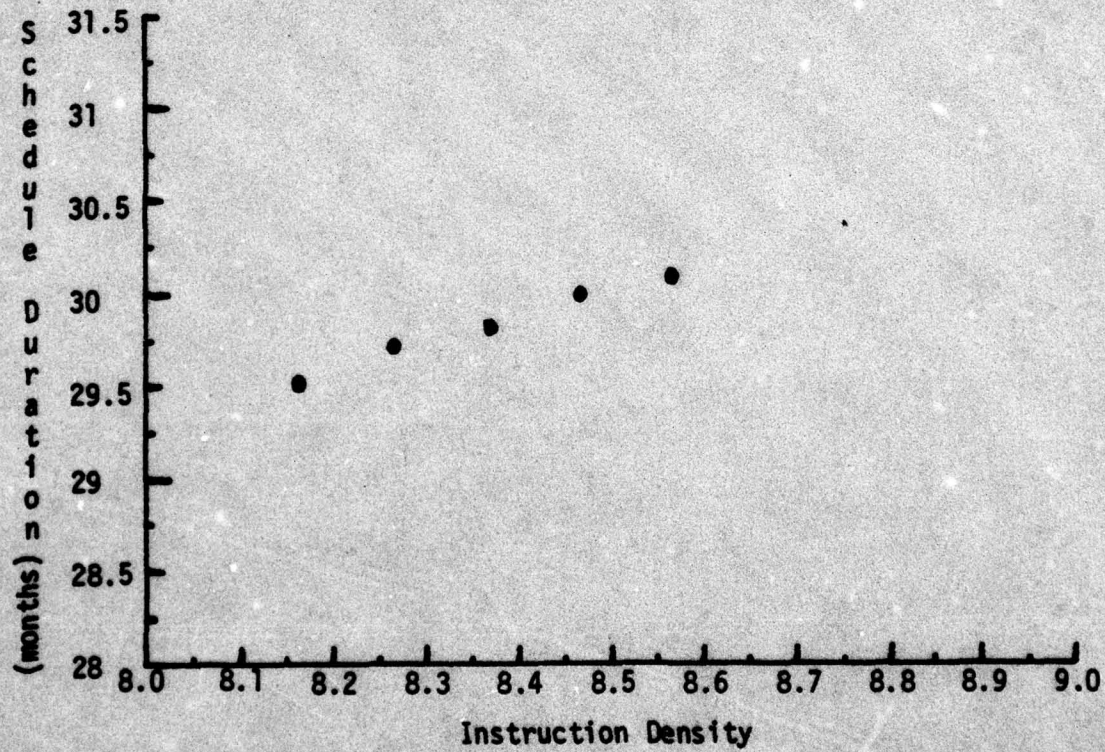


Figure 13. Schedule Duration vs. Instruction Density - System E

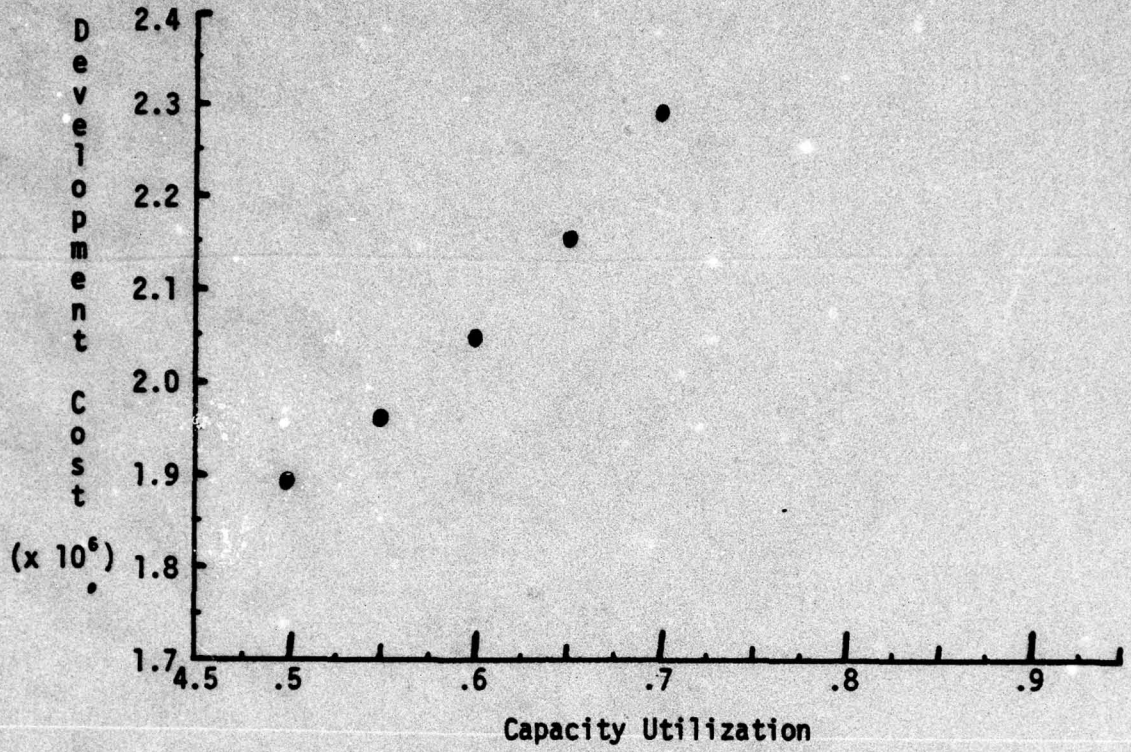


Figure 14. Development Cost vs. Capacity Utilization - System A

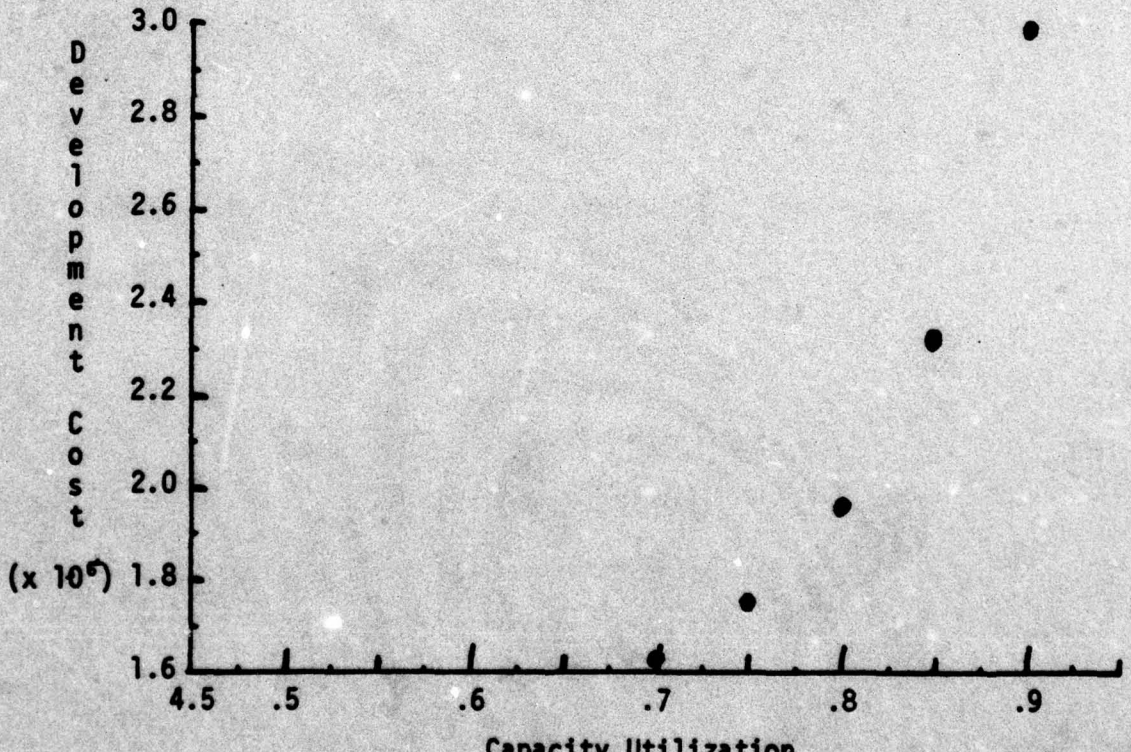


Figure 15. Development Cost vs. Capacity Utilization - System E

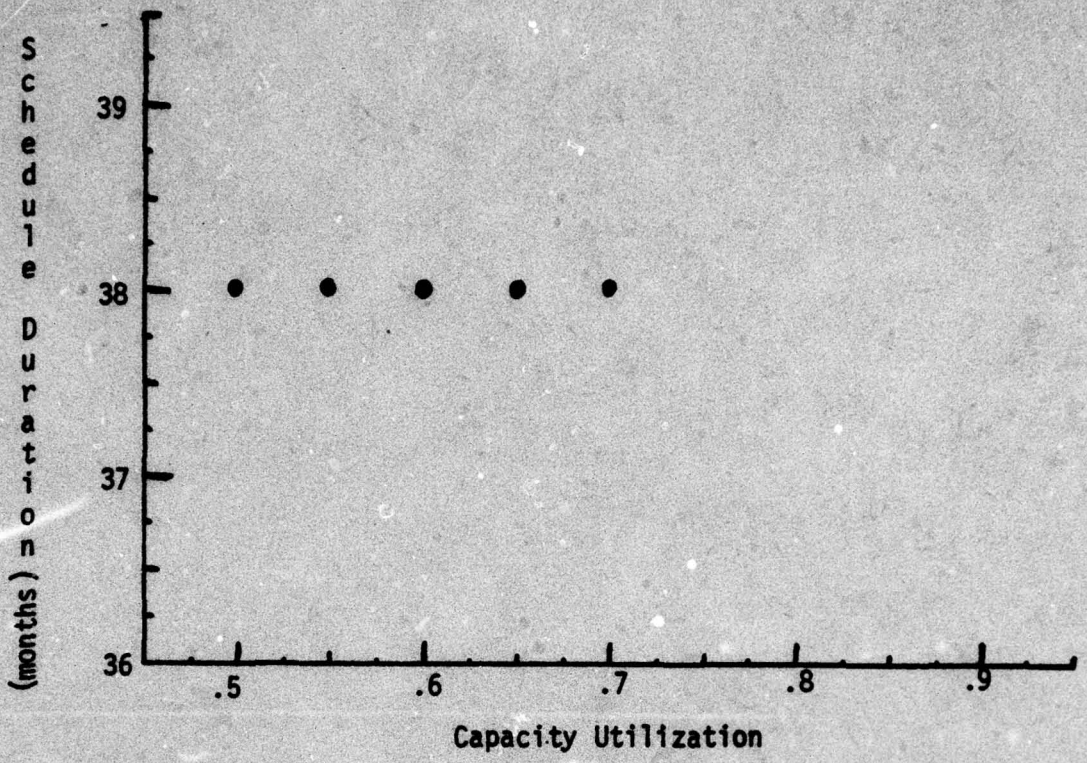


Figure 16. Schedule Duration vs. Capacity Utilization - System A

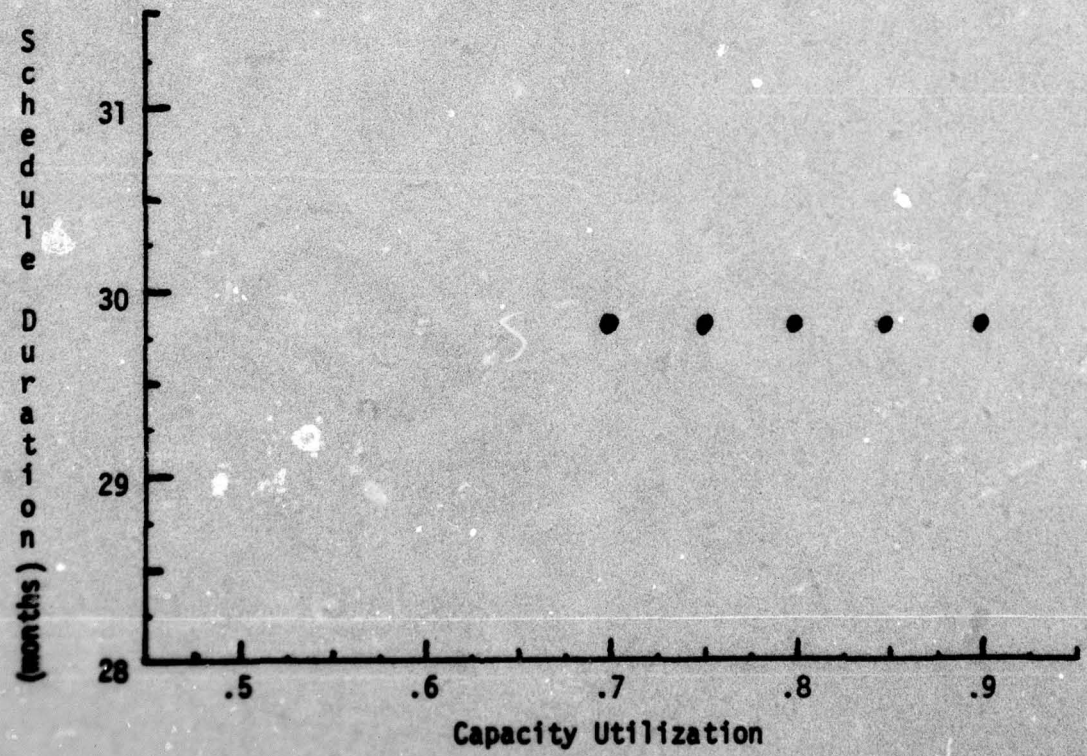


Figure 17. Schedule Duration vs. Capacity Utilization - System E

figures, it is apparent that the only relationship that is significantly nonlinear is the effect of UTIL on software development cost. This is especially true of System E, where the baseline value of UTIL was 0.8, a rather high value. A severe cost penalty is associated with the increased effort necessary to "force" the system to fit into the limited capacity available. The exponential shape of this cost curve is not unreasonable considering the increasing restrictions the development team must overcome as the capacity limitations become more severe.

It is interesting to note that the PRICE S runs showed no effect at all of UTIL on schedule duration. Intuitively, this does not seem to be correct. The problems associated with capacity limitations are not generally solved by hiring more people, or buying equipment. They are usually solved through a process of continuing refinement that simply takes time. The failure of PRICE S to reflect this may indicate an oversight or an error in the particular version of the PRICE S computer program used.

The effects of variations of the "soft" PRICE S parameters on cost and schedule were examined using multivariate linear regression and three way ANOVA techniques. Table VIII summarizes both the regressions, and the ANOVA results. It is obvious from these two tables that, in the neighborhood of the baseline parameters, the "soft" parameters effects are also quite linear. In addition, no evidence of significant interactions between the input variables were found. Thus, at least within the neighborhood of a particular set of PRICE S parameters, these parameters may be treated as an orthogonal set of independent parameters.

The results of the sensitivity analysis seem quite reasonable, with one qualification. The lack of any dependency of schedule duration on

TABLE VIII

Sensitivity Analysis Results

Regression Models: $\Delta COST = a_0 + a_1 \Delta SDCPLX + a_2 \Delta ENCPLX + a_3 \Delta PLTFM$

$\Delta SCHED = b_0 + b_1 \Delta SDCPLX + b_2 \Delta ENCPLX + a_3 \Delta PLTFM$

SYSTEM A			SYSTEM E		
i	a_i	b_i	i	a_i	b_i
0	6	.01	0	6	.26
1	1001	1.98	1	951	-.95
2	1251	19.13	2	1301	20.30
3	1801	0.0	3	1706	0.0

ANALYSIS OF VARIANCE (ANOVA)

	COST	
	SYSTEM A	SYSTEM E
Total Sum of Squares (SS)	34,181,443	31,691,305
Less: SS due to grand mean	33,714,366	31,248,465
SS due to all other effects	467,077	442,840
Less: SS due to ΔSDCPLX	80,200	72,390
	386,877	370,450
Less: SS due to ΔENCPLX	125,250	135,460
	261,627	234,990
Less: SS due to ΔPLTFM	259,560	232,903
SS due to interactions, errors, and all other effects	2,067	2,087

	SCHEDULE	
	SYSTEM A	SYSTEM E
Total Sum of Squares (SS)	11,586.14	7,269.54
Less: SS due to grand mean	11,556.56	7,236.05
SS due to all other effects	29.58	33.49
Less: SS due to ΔSDCPLX	0.31	0.07
	29.27	33.42
Less: SS due to ΔENCPLX	29.26	32.97
SS due to ΔPLTFM, interactions, errors and all other effects	0.01	0.45

PLTFM lacks intuitive appeal. Increases in specification requirements imply more detailed test plans and procedures, more extensive "debugging" efforts, and more testing. It would be expected that all of these factors would tend to lengthen the development schedule.

V. SUMMARY AND CONCLUSIONS

The stated purpose of this research was to investigate ways of gathering and using descriptive data for the purpose of making preliminary software cost estimates. Such data is necessary, because the analysts who make preliminary cost estimates for software at ASD rely heavily on hisotrical analogy. Considering the state-of-the-art of software cost estimation, it appears that this will continue to be the case for the forseable future.

The purpose of the research was achieved through the accomplishment of two specific objectives. These were to develop a methodology for gathering descriptive data on software systems, and to use historical ASD software acquisition data to calibrate the PRICE S model. These objectives were related in that the data necessary to achieve the second objective was collected using the methodology developed in achieving the first. This allowed the researcher to evaluate the effectiveness of the methodology in terms of the usefulness of the data collected.

Conclusions

The specific conclusions which may be drawn from this research may be grouped under two classifications. First, conclusions relating to the data collection methodology and its potential usefulness are presented. Finally, the conclusions relating to the PRICE S model are stated.

Conclusions Relating to the Methodology. It was found that the methodology described above could be used to objectively measure some of the distinctive characteristics of software systems.

It was found that the data collected using the methodology could be used for cost estimating purposes.

It was found that both the statistical and subjective information collected provided meaningful insights into specific software systems. These insights may be used to draw analogies between systems for cost estimating purposes.

It was found that the sequential sampling technique used to gather statistical data about software systems did substantially lower the amount of work required to calculate the desired descriptive parameters.

No way could be found to eliminate the need to count individual machine instructions within the modules sampled.

It was found that an attempt should have been made to determine objective values for "fraction of total system capacity utilized." This data might have been collected from existing correspondence or formal reports. In some cases, researchers may attempt to estimate this parameter independently by analyzing system "load maps" or test reports. Future researchers should modify their data collection efforts to include the additional effort.

In general, it was found that while the data collection methodology worked, it was cumbersome. Two man-weeks or more are required to gather data on a single avionics software system of about 300 modules using the methodology described above. While this may be feasible for systems in this size range, the researcher cannot recommend that this methodology be applied to large systems containing thousands of modules.

Conclusions Relating to PRICE S. The following conclusions may be stated with respect to the RCA PRICE S software cost estimation model:

It was found that historical ASD software cost data is not incompatible with the PRICE S model.

It was found that most of the PRICE S input parameters are amenable to objective measurement. The exceptions to this finding are "system design complexity" (SDCPLX) and "engineering complexity" (ENCPLX), which must be related to objective reality through empirical studies (calibration).

Finally, it was found that, in practice, the PRICE S model is quite flexible. This is both a strength and a weakness. The model seems to have the flexibility to be useful in a wide range of situations. This same flexibility, however, makes it doubtful that the model should be used independently of other cost estimating techniques. The assumptions reflected in the input parameters must be continually questioned and compared to objective reality if the estimates derived from the model are to be relied upon.

BIBLIOGRAPHY

Aaron, J. D., Estimating Resources for Large Programming Systems, International Business Machines Corp., Gaithersburg, Maryland, 1969.

Boehm, Barry W., "The High Cost of Software" Practical Strategies for Developing Large Scale Software Systems, E. Horowitz, Ed., Addison-Wesley Publishing Co., Reading, MA, 1975, 3-14.

Brooks, Fredrick P., The Mythical Man-Month, Essays on Software Engineering, Addison-Wesley Publishing Co., Reading, MA, 1975.

Clapp, J. A., A Review of Software Cost Estimating Methods, The MITRE Corporation, Bedford, MA, 1976 (AD - A029748).

Conover, W. J., Practical Nonparametric Statistics, John Wiley and Sons, Inc., New York, 1971. .

Department of the Air Force, Air Force Regulation 800-14, Vol II, "Computer Resources Acquisition and Support" HQ USAF, Washington, D.C. 1974.

Draper, N. R. and H. Smith, Applied Regression Analysis, John Wiley and Sons, Inc., New York, 1966.

Farr, Leonard and Burt Nanus, Factors that Affect the Cost of Computer Programming, System Development Corporation, Santa Monica, California, 1964. (AD - 447329)

Freiman, Frank, "Lectures and personal conversations", Moorestown, New Jersey, 8 and 9 May, 1977.

Freund, John E., Mathematical Statistics, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.

Gansler, Jacques S., "Software Management in the Department of Defense" Abridged Proceedings from the Software Management Conference, American Institute of Aeronautics and Astronautics, Los Angeles, CA 1976.

Graver, C. A., et. al. Cost Reporting Elements and Activity Cost Tradeoffs for Defense System Software, General Research Corporation, Santa Barbara, California, 1977.

Herd, James H., et al. Software Cost Estimation Study, Doty Associates, Inc., 1977.

Nelson, Edward A., Methods of Obtaining Estimates of Computer Programming Costs: A Taxonomy, System Development Corporation, Santa Monica, California, 1966.

Peterson, J. B., System Software Acquisition, Unpublished Lecture, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, 1977.

Putnam, Lawrence H., A General Solution to the Software Sizing and Estimating Problem, Unpublished paper, 1977.

_____, Preliminary - May 1977, Reference Manual, PRICE Software Model RCA PRICE Systems, Inc., Moorestown, New Jersey, 1977.

Smith, Ronald L. Structured Programming Series, Volume XI, Estimating Software Project Resource Requirements, Final Report. International Business Machines Corp., Gaithersburg, Maryland, 1976. (AD - A016416)

_____, Webster's Seventh New Collegiate Dictionary, G. and C. Merriam Company, Springfield, Massachusetts, 1965.

Weinwurm, G. F. and H. J. Zuvorski. Research into the Management of Computer Programming: A Transitional Analysis of Cost Estimation Techniques. System Development Corporation, Santa Monica, California, 1965. (AD - 631289).

Wolverton, Ray W., "The Cost of Developing Large-scale Software" IEEE Transactions on Computers, June 1974, Vol C-23:615-636.

APPENDIX A

A Glossary of PRICE S Terminology

APPENDIX A

A Glossary of PRICE S Terminology

ENCPLX	(Engineering complexity) Relates difficulty of task to time schedule for completion.
FCONST	Empirical factor describing the extent of echeloning (skewness) of the hierarchical, functional structure.
FUNCT	Total number of functional modules.
IDENS	(Instruction density) Describes the mixture of instruction types. The weighting factor used to classify application types.
INSPF	Average number of machine level instructions per functional module.
LEVEL	Mean hierarchical level. Average tree structure level at which functional modules are defined.
MIX	(Application mix) Describes software profile in terms of total code in each of seven application types: mathematical applications, string manipulation, system operation, data storage and retrieval, on-line communications, real-time command and control, and interactive.
SDCPLX	(System design complexity) Relates scope of work to the engineering group doing the work.
TNINST	Total number of machine-level instructions.
UTIL	Fraction of total system capacity utilized.

(RCA PRICE SYSTEMS, Inc., 1977)

APPENDIX B

The Initial Interview Format

INITIAL INTERVIEW WORKSHEET (Page 1 of 4)

System Identifier: ____ Date: _____

System Name: _____

Individuals to contact:

NAME	RANK	OFFICE SYMBOL	OFFICE PHONE

QUALIFICATION QUESTIONS

VEHICLE TYPE: _____ SINGLE COMPUTER: YES NO

ASSEMBLY LANGUAGE: YES NO INHERENTLY MODULAR: YES NO

SOFTWARE DEVELOPMENT COMPLETE TO AIR FORCE ACCEPTANCE: YES NO

TYPE OF DOCUMENTATION AVAILABLE: PART I OR EQUIVALENT

PART II OR EQUIVALENT

CODE

CPR'S OR EQUIVALENT

REQUIRED SAFEGUARDS/RESTRICTIONS ON DATA USAGE: _____

_____

INTERVIEWER COMMENTS: _____

_____

_____

_____

_____

_____

System Identifier: ____ Date: _____

QUALITATIVE QUESTIONS

Is this a stand-alone system, or part of a distributed processing software system? _____

_____

If part of a distributed system, does it perform a single (or homogeneous group of) function(s)? _____

_____

Does this system share the host computer with another system?

_____

_____

Describe the pertinent characteristics of the host computer.

Core Size: _____ Word Size: _____ KOPS: _____

Other (SPECIFY): _____

_____

Was the development schedule either unusually long or short?

_____

_____

_____

Was the group which developed the software system exceptionally well qualified or inexperienced with this type of software system?

_____

_____

AD-A046 808 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
A PRELIMINARY CALIBRATION OF THE RCA PRICE S SOFTWARE COST ESTI--ETC(U)
SEP 77 J SCHNEIDER

UNCLASSIFIED AFIT/GSM/SM/77S-15 NL

2 OF 2
AD A046808

END
DATE FILMED
12-77
DDC

System Identifier: ____ Date: _____

QUALITATIVE QUESTIONS (CONT)

Did the developer use any of the new software development methodologies? If so, which ones? Did he have previous experience with them? _____

_____

Were the system specification requirements either unusually stringent or lax? If so, in what way? _____

_____

_____

_____

Did the development either approach or extend the state-of-the-art in any way? If so, how? _____

_____

_____

Was a significant portion of the design completed before work officially began on the contract? If so, describe the situation.

_____

_____

_____

INITIAL INTERVIEW WORKSHEET (Page 4 of 4)

System Identifier: ____ Date: _____

SYSTEM LEVEL DATA

Primary purpose/function performed: _____

_____

_____

Major sub-functions performed: _____

_____

_____

Percent of processor capacity utilized: Memory _____

Processing Time _____ I/O Channels _____

Others above 50% (Specify): _____

Total number of devices, by type, that interact with/are controlled by the software system (Identify unique devices). _____

_____

_____

_____

_____

What COST/SCHEDULE/MANPOWER LOADING information is available?

_____

_____

_____

APPENDIX C

Derivation of the Sequential Sampling Equation

APPENDIX C

Derivation of the Sequential Sampling Equation

Objective

Determine the sample size n required to assure that the bounds of a 95 percent confidence interval for mean module weight $\bar{w}$ lie within ten percent of $\bar{w}$.

Mathematically, this may be stated:

$$P(|\bar{w} - \mu_{\bar{w}}| \leq 0.1\bar{w}) \geq 0.95$$

where: $\bar{w}$ = The sample mean of the random variable "module weight."

$\mu_{\bar{w}}$ = The true mean module weight of the population.

P = The probability function.

Given: (1) A finite population, the software system, containing N elements (modules), each of which is equally likely to be chosen in the sampling process.

(2) Each module has a unique value of a random variable called weight, w associated with it. Weight is calculated by multiplying the number of instructions in the module by a weighting factor which is determined by the application type of the module.

Assumptions

The sample size n is large enough for the Central Limit Theorem to be invoked. This assumption is discussed below, following the derivation itself.

Derivation

Assume that a sample of n modules is drawn, without replacement, from a software system containing a total of N modules. Let:

$$Y_n = \sum_{i=1}^{n} w_i = n\bar{w} \qquad \text{(a)}$$

$$\mu_n = E(Y_n) = n\mu_w \qquad \text{(b)}$$

$$\sigma_n^2 = VAR(Y_n) = VAR(\sum_{i=1}^{n} w_i) = n^2 \, var(\bar{w}) \qquad \text{(c)}$$

Where: w_i = Weight of the i^{th} module

n = Sample size

$\bar{w}$ = The sample mean of weight

μ_n = The population mean of Y_n

μ_w = The population mean of the w_i

σ_n^2 = The variance of Y_n

Then, by the Central Limit Theorem, the distribution of the random variable

$$z = \frac{Y_n - \mu_n}{\sigma_n} \qquad \text{(d)}$$

approaches the standard normal distribution as n approaches infinity. (Conover, 1971:53-54)

Substituting equations (a), (b), and (c) into equation (d) yields:

$$z = \frac{n\bar{w} - n\mu_w}{n \; VAR(\bar{w})^{\frac{1}{2}}} = \frac{\bar{w} - \mu_w}{\sigma_{\bar{w}}} \qquad \text{(e)}$$

Where: $\sigma_{\bar{w}} = (VAR(\bar{w}))^{\frac{1}{2}}$

Now, according to Freund, if a random sample is drawn, without replacement, from a finite population, then the variance of the sample mean is given by:

$$VAR(\bar{w}) = \frac{\sigma_w^2}{n} \frac{(N-n)}{(N-1)} \qquad \text{(Freund, 1971:204)} \qquad \text{(f)}$$

If n is sufficiently large for the Central Limit Theorem to be invoked, then the upper bound of a 95 percent confidence interval is defined by the condition:

$$z_u = 1.96$$

Where: z_u = The upper bound of the confidence interval.

Since the normal distribution is symetric about the mean, it is sufficient to examine the upper limit only. Analogous results follow for the lower bound.

Finally, the requirement that the bounds of the confidence interval lie within ten percent of the sample mean implies that at the upper boundary, the condition:

$$\bar{w} - \mu_w = 0.1\,\bar{w} \qquad \text{(h)}$$

must hold. Again, because of symmetry, it is not necessary to consider the lower bound separately.

Applying the boundary conditions of equations (g) and (h) to equation (e), and then substituting equation (f) into equation (e) yields:

$$\frac{0.1\,\bar{w}}{\sigma_w \sqrt{\frac{N-n}{n(N-1)}}} = 1.96 \qquad (i)$$

This equation defines the conditions necessary for the upper bound of a 95 percent confidence interval to lie precisely (.1 $\bar{w}$) from $\bar{w}$. By squaring both sides of equation (i), solving for n, and replacing w with the unbiased estimator σ_w it follows that:

$$n = \frac{N}{\frac{w^2}{384}\frac{(N-1)}{S_w^2}} \qquad (j)$$

This is the equation to be used in determining the sample size required for the statistical data collection described in Chapter III above.

Use of the Central Limit Theorem

Conover states that the Central Limit Theorem may be invoked when drawing without replacement from a finite population provided that the population size N is a least twice the sample size n. (Conover, 1971: 54) There is no discussion of whether the theorem may be applied for larger sample sizes than (N/2). Since both systems studied required sample sizes that exceeded this restriction (based on equation (j)) the applicability of the Central Limit Theorem must be addressed.

There is a very definite relationship between the sum of n random variables drawn without replacement from a finite population, and the sum of the (N-n) variables which were not drawn. (Failure to be selected may be considered a form of selection.)

In fact, it may be shown that:

$$\mu_{r,n} = (-1)^r \mu_{r,(N-n)}$$

Where: $\mu_{r,n}$ = the r^{th} moment about the mean of the sum of the n variables drawn, and

$\mu_{r,(N-n)}$ = the r^{th} moment about the mean of the (N-n) variables not drawn.

From the above, it may be inferred by symmetry that the distribution of the sum of a sample of size (N-n) approximates the normal distribution precisely as well as that of a sample of size n. Thus, it would seem that the operative constraint on the invocation of the Central Limit Theorem is that both (N-n) and n must be "large." This condition is less restrictive than the "50 percent rule" cited above. For both of the systems sampled, both (N-n) and n exceeded 100, therefore the Central Limit Theorem was properly invoked.

APPENDIX D

System Descriptions

SOFTWARE DEVELOPMENT DATA

SYSTEM NAME: SYSTEM A DATE: 11 SEPT 1977

PRIMARY FUNCTION: RADAR JAMMING

MAJOR SUB-FUNCTIONS: PROCESS RADAR PULSES, IDENTIFY THREATS, ALLOCATE JAMMERS, DISPLAY STATUS, OPERATOR INTERFACE, TEST

SYSTEM ENVIRONMENTAL DATA

PLATFORM

GROUND (FIXED OR MOBILE)
AVIONICS (CIVIL OR MILITARY)
SPACE (MANNED OR UNMANNED)

TARGET COMPUTER(S)

NUMBER	MODEL	WORD SIZE	MEMORY SIZE	PER CENT CAPACITY UTILIZED
1	4-PI	32	16K	50-60
1	LC-4516	16	8K	50-60

SYSTEM DESCRIPTION

TOTAL INSTRUCTIONS SOURCE	TOTAL INSTRUCTIONS OBJECT	LANGUAGE	NUMBER OF MODULES	MEAN HIERARCHICAL LEVEL
UNK	19,000	ASSEMBLY	307	2.6

INSTRUCTION MIX

MATH	STRING	SYSTEM	DATA	ON-LINE	REAL-TIME	INTERACTIVE
.00	.18	.16	.05	.16	.13	.32

HARDWARE INTERFACE DEVICES		ON-LINE	REAL-TIME	INTERACTIVE
	NO. MODELS	0	2	1	1
	TOTAL	0	2	1	1

DEVELOPMENT INFORMATION

TOTAL COST: 1,054,000 ACCOUNTING ASSUMPTIONS/PROBLEMS: Single software development account, plus small IV&V contract. Systems engineering costs not included. Burden in, G&A out.

START DATE: JAN 75 END DATE: MAR 78 (EST)

ESTIMATED NORMAL TIME FOR DEVELOPMENT: 38 Months

QUALIFICATIONS OF WORK GROUP: HIGH 7 6 5 4 3 2 1 LOW

PAY SCALE OF WORK GROUP: HIGH 7 6 5 4 3 2 1 LOW

ASSUMPTIONS/LIMITATIONS/COMMENTS: DEVELOPMENT NOT COMPLETED WHEN DATA TAKEN. FORMAL QUALIFICATION TESTING WAS WELL UNDERWAY WITH NO APPARENT PROBLEMS. LACK OF SYS ENG COSTS.

SOFTWARE DEVELOPMENT DATA

SYSTEM NAME: SYSTEM E DATE: 11 SEPT 1977

PRIMARY FUNCTION: NAVIGATE AND CONTROL A GUIDED MUNITION

MAJOR SUB-FUNCTIONS: DETERMINE WEAPON LOCATION, CALC GUIDANCE ERROR, GUIDE. CMDS., PITCHOVER CALCS, CALC LAUNCH WINDOWS, INITIATE WEAPON RELEASE.

SYSTEM ENVIRONMENTAL DATA

PLATFORM	TARGET COMPUTER(S)				
	NUMBER	MODEL	WORD SIZE	MEMORY SIZE	PER CENT CAPACITY UTILIZED
GROUND (FIXED OR MOBILE)					
AVIONICS (CIVIL OR MILITARY)					
SPACE (MANNED OR UNMANNED)	1		32	16K	HIGH

SYSTEM DESCRIPTION

TOTAL INSTRUCTIONS SOURCE	TOTAL INSTRUCTIONS OBJECT	LANGUAGE	NUMBER OF MODULES	MEAN HIERARCHICAL LEVEL
UNK	16,000	ASSEMBLY	297	4.0

INSTRUCTION MIX

MATH	STRING	SYSTEM	DATA	ON-LINE	REAL-TIME	INTERACTIVE
.12	.01	.11	.00	.03	.46	.27

HARDWARE INTERFACE DEVICES		DATA	ON-LINE	REAL-TIME	INTERACTIVE
	NO. MODELS	0	4	4	1
	TOTAL	0	5	VARIES	1

DEVELOPMENT INFORMATION

TOTAL COST: 1,800,000 ACCOUNTING ASSUMPTIONS/PROBLEMS: PRIME TASK ON SINGLE CONTRACT. COST OF OTHER TASKS BACKED OUT. BURDEN INCLUDED, G&A EXCLUDED

START DATE: APRIL 1974 END DATE: DEC 1976

ESTIMATED NORMAL TIME FOR DEVELOPMENT: 30 MONTHS

QUALIFICATIONS OF WORK GROUP: HIGH 7 6 5 4 3 2 1 LOW

PAY SCALE OF WORK GROUP: HIGH 7 6 5 4 3 2 1 LOW

ASSUMPTIONS/LIMITATIONS/COMMENTS: UTILIZATION OF CAPACITY NOT MEASURED. THERE WAS PRESSURE TO SHORTEN DEVELOPMENT SCHEDULE.

VITA

Captain John Schneider IV was born on October 31, 1945 in Greensboro, North Carolina, but was raised in Cambridge, Maryland. He attended the University of Maryland and received a Bachelor of Science degree in Physics from that institution in 1968. Having participated in the Reserve Officer Training Corps program as an undergraduate, he entered the Air Force as a second lieutenant upon graduation.

From 1968 through 1971, Captain Schneider was assigned to the 1 Aerospace Control Squadron (ADC) in Colorado Springs, Colorado, where he was responsible for developing and implementing techniques for tracking and predicting the position of highly eccentric and geo-stationary earth satellites. After a short tour in Thailand, Captain Schneider was reassigned to the Colorado Springs area in late 1972. For the next few years, he worked as a computer programmer on several, astrodynamic computer programs for the NORAD Space Defense Center (SDC) and acted as a technical consultant on the software system for the Space Computational Center (the follow-on facility for the SDC). During the year ending in June, 1976, Captain Schneider organized and managed a group of military and civilian computer programmers who were responsible for developing a sub-system of astordynamic command and control programs for the Space Computation Center.

Captain Schneider is currently assigned to the School of Engineering Air Force Institute of Technology, where he is pursuing a Master of Science degree in Systems Management.

Captain Schneider is married to the former Miss Susan Lake of East New Market, Maryland. They have two children, Laura and John.

Permanent address: 107 Mill Street
Cambridge, Maryland 21613

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Master's Thesis

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. AFIT/GSM/SM/77S-15	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A PRELIMINARY CALIBRATION OF THE RCA PRICE S SOFTWARE COST ESTIMATION MODEL,	5. TYPE OF REPORT & PERIOD COVERED: M.S. Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John Schneider, IV Captain USAF	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433	12. REPORT DATE: Sep 77 13. NUMBER OF PAGES: 110
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES: Approved for public release; IAW AFR 190-17

JERRAL F. GUESS, Capt, USAF
Director of Information

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Cost Estimates
Data Acquisition
Computer Programming

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Each year, the Department of Defense spends more than three billion dollars on computer software, yet software managers are notoriously unable to predict the cost of software development projects. This is especially true of preliminary cost estimates made during the formative stages of a project. Even when parametric relationships are used, such estimates depend heavily on anology with previously developed systems. The purpose of this research is to investigate ways of gathering and using descriptive data for the purpose of making preliminary software cost estimates. A methodology for the collection of

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

A22225

descriptive information on software systems was developed and used to describe several avionics software systems. The data thus gathered was then used to "calibrate" the PRICE S software cost estimation model by relating particular values of several "subjective" PRICE S input parameters to the observed software systems could be objectively measured, and that the PRICE S model is not incompatible with avionics software systems developed for the Aeronautical Systems Division of Air Force Systems Command.